

MoteLife

Frank J. Murphy

A dissertation submitted to the University of Dublin,
in partial fulfilment of the requirements for the degree of
Master of Science in Computer Science

2007

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: _____

Frank J. Murphy

September 2007

Permission to lend and/or copy

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

Signed: _____

Frank J. Murphy

September 2007

Acknowledgements

Firstly I wish to thank my supervisor, Dr Ciaran McGoldrick, for providing the right framework enabling this work to be done. He directed me to ask the right questions and kept my focus on the project. I would also like to thank Ricardo Carbajo, whose deep knowledge of TinyOS saved many hours of searching through documentation and discovering things the hard way.

I am also indebted to Cahill Printers Limited, my employer during this period, who allowed me very flexible working hours making my studies possible. I would like to thank all the lecturers who gave me new perspectives on aspects of technology and my classmates who made the whole experience so enjoyable.

I want to thank my family and friends, who remain close despite my infrequent appearances over the last year. Finally, I thank my wife for her constant understanding and encouragement. Her sacrifices, even during pregnancy, allowed me devote my attention to this work.

Abstract

The provision of an appropriate power source is one of the key design requirements for modern mobile electronic devices. This requirement is particularly compelling in the case of Wireless Sensor Network nodes (motes) as such devices may not be physically accessible after deployment. Thus it is of critical importance that power usage and consumption be very tightly controlled in such an environment. In this thesis, an examination of battery discharge is performed on Mica2 motes, under various operating conditions. Predicted current consumption, estimated using duty cycles and power ratings, is compared with measured current consumption and actual discharge time. A TinyOS component is created, providing a framework for enabling battery discharge and power consumption self-awareness within motes. The use of a rule based decision engine allows a device to manage battery drain and respond to changes in power consumption.

Table of Contents

1 Introduction.....	1
2 Background.....	4
2.1 Mote Hardware.....	4
2.2 Mote Development Environment.....	10
2.3 Literature Review.....	11
2.3.1 Avoiding Extra Transmissions.....	11
2.3.2 Reducing Radio Cycles.....	13
2.3.3 Power aware routing.....	15
2.3.4 Adjusting Transmit Power.....	17
3 Work Undertaken.....	19
3.1 Battery Drain.....	20
3.2 Measurements of Battery Drain.....	22
3.2.1 KillMote User Interface.....	23
3.2.2 Experiment 1: No Operation.....	24
3.2.3 Experiment 2: Continuous Signal Transmission.....	26
3.2.4 Experiment 3: Continuous Signal Reception.....	29
3.2.5 Experiment 4: LEDs On.....	31
3.2.6 Experiment 5: Continuous CPU Activity.....	33
3.2.7 Experiment 6: Continuous Voltage Sensing.....	35
3.2.8 Experiment 7: Continuous Light Sensing.....	36
3.2.9 Overall Result.....	38
3.2.10 Current Consumption Estimation.....	38
3.3 Predicting Remaining Battery Life.....	39
3.3.1 Experiment 1: No Operation.....	41
3.3.2 Experiment 2: Continuous Signal Transmission Prediction.....	42
3.3.3 Experiment 3: Continuous Signal Reception Prediction.....	43
3.3.4 Experiment 4: LEDs Always On Prediction.....	44
3.3.5 Experiment 5: Continuous CPU Activity Prediction.....	45
3.3.6 Experiment 7: Continuous Light Sensing.....	46

4 Implementation Details.....	47
4.1 Self Preservation Module.....	47
4.1.1 SelfP Interface.....	48
4.2 KillMote with SelfP.....	53
4.3 Problems Encountered.....	54
5 Conclusions and Future Work.....	56
References.....	58

1 Introduction

The term “mote” is used to describe small computing devices that usually have both wireless communications and sensing capabilities. The range of sensors that can be fitted to motes include accelerometers, magnetometers, thermometers, Ground Positioning System (GPS) sensors, microphones, photo detectors and voltmeters for the power source. They are fitted with a processor that has access to both volatile and non-volatile memory. They can form ad-hoc networks and pass information between themselves and also back to a base station. The base station is typically a more powerful device with more processing power, possibly mains supply and greater storage capacity. A Personal Computer (PC) or laptop can be used as a base station. It may have access to the Internet or other networks that can be used to send information back to a central point after being analysed and summarised. The goal is to build such devices within a cubic millimetre so they can truly resemble smart dust [16] and they are indeed approaching this size. Most motes today are powered with batteries, but as they get smaller it becomes more feasible to power them solely from photovoltaic cells or other power producing sources. The diminishing size and increasing capabilities of these devices is enabling a new generation of Wireless Sensor Network (WSN) applications. Vehicle detection is easily implemented using motes. Such motes deployed in a WSN can deduce the volume of traffic, speed and even the type of vehicle [10]. The military have shown a keen interest in using WSN on the battlefield. An aircraft can drop a huge number of these devices over an area and they can relay accurate and up-to-date information back to headquarters showing movements of people and vehicles or sudden changes in environmental parameters [24, 17]. Security is another area where a WSN can be deployed. It is easy to imagine motes being placed on doors and windows in a home. They could then notify the owner when movement is detected. A home WSN could be used to ensure that the temperature in an area remains within limits, or to alert someone if the freezer gets too warm. Structural health monitoring of buildings, bridges, aircraft, ships and vehicles are other areas for WSN deployment [56]. In the future, when devices are sufficiently small and robust, they may be integrated into the fabric of structures, but until then they are fixed to the surface at strategic points in order to capture data. WSNs are being used increasingly for environment and habitat monitoring [43] where a large number of distributed motes can provide a huge amount of data, enabling a

level of analysis not previously possible. Motes can be attached to animals, as in the ZebraNet project [30] or be used in inaccessible areas such as erupting volcanoes [51] or underwater [2]. When motes are fitted with legs, wheels or wings, swarms of autonomous micro robots give rise to many more types of application.

One problem shared by the motes in all these applications is the relatively short lifespan of their power source. If power is not managed, the mote will die and the WSN will cease to be useful. If a mote finds itself bridging two parts of a WSN that would otherwise not be connected, it is likely to suffer from having to pass many messages from one side to the other. Such a node is likely to have its power supply exhausted before most of the other nodes. Such configurations give rise to network partitioning and dead areas in the WSN (until the batteries are replaced). Given the applications described, it is not always feasible or economically viable to replace batteries in motes. In military applications, the motes may be located in areas under enemy control. In environment and structural monitoring, motes may be placed in locations that are hazardous or hard to reach.

Applications in security, structural monitoring and the military, attract a wide variety of motivated attackers including criminals, terrorists and hostile nations. Motes in such networks are constrained by resources making public key cryptography very expensive and even fast symmetric key ciphers must be used sparingly. Each bit transmitted is equivalent in power loss to executing up to 1,000 instructions, making any expansion of messages due to security mechanisms unacceptable. Motes are particularly vulnerable to Denial of Service (DoS) attacks, as any incoming signal, which triggers a response from the node, will use up precious power. The easiest way to render an enemy node useless is to repeatedly send it signals that trigger return signals. Using multiple nodes to set up a distributed DoS attack would disable such unprotected nodes quite easily. Without encryption, such networks are also susceptible to eavesdropping, spoofing, insertion, deletion and replay of messages. The level of security required depends on the application, the likelihood of attack and the likely strength of potential attackers.

A method for motes is proposed to predict how long they will continue to operate in the current environment. Having estimated the remaining lifetime, a flexible rule-based approach is used for motes to modify their actions in response to signals, so that they may extend their lifetime, or fulfill their primary objective. A framework for

motes is provided to expose factors that affect their lifecycle. This is done through a self-preservation module, which a node uses to protect itself.

Any particular WSN is put in place with an objective in mind. For example, a WSN may be installed to record temperature over time and send the results back to a base station. In such data gathering networks motes are responsible for measuring and recording data and also handle routing messages back to the base station. In this application a mote that finds itself close to exhaustion, may find that if it continues to route signals for other nodes back to the base station, it will be unable to send its own data back to the base station. The node can then signal to local nodes that it is no longer routing messages and continue to record the temperature until it is time to send the recorded data to the base station, thus saving valuable data. In a military environment it may be more important to ensure that a message warning of vehicle movement is passed back to headquarters, even if it will result in some inability to monitor the environment. The proposed framework also allows motes to detect abnormal rises in activity. Such a rise in activity could be due to a DoS or other attack and a mote may use this information to take evasive action. Significantly, having knowledge of its remaining lifetime also allows a mote to participate in Active Queue Management (AQM) schemes [34]. In network architectures, congestion based pricing supports Quality of Service (QoS) and resource allocation [28, 29]. In a WSN it is appropriate that the cost of using a mote with limited energy reserves should be much higher than routing through a mote with ample reserves. WSN pricing is likely to be based on available energy reserves. Most WSNs in use today are purpose built with tailored messaging and most abstractions are specialized and application specific [35]. As WSNs evolve and abstractions become more common, it may become feasible for WSNs to interoperate and negotiate for the use of resources. The self-preservation module giving knowledge of current usage and predicted lifetime is a key enabler for AQM participation.

2 Background

This section starts by examining mote hardware, with particular reference to the power supply and current consumption. The most common software development platform is then examined before going on to review the literature in connection with power consumption in wireless sensor networks. This sets the background for conducting tests and developing a framework for motes to predict and preserve their lifetime.

2.1 Mote Hardware

The most common devices in use today are the range of motes produced by Crossbow Technology Inc. Most academic papers, including this one, focus on the Mica 2 (Figure 1) from Crossbow, described in the Mica2 datasheet [12].



Figure 1: Mica2 Mote

This mote measures 58cm x 32cm and is therefore closer to the size of a matchbox than a spec of dust. The processor in a Mica2 is an ATmega128L running at 4MHz. This is a CMOS 8-bit microcontroller from Atmel, based on RISC (Reduced Instruction Set Computer) architecture [1]. This executes powerful specialised instructions in a single clock cycle and is optimised for low power consumption. The datasheet [12] shows that the system has 128 KB bytes of program instruction memory, 4 KB of RAM and 512 KB of flash memory. Three Light Emitting Diodes (LEDs) together with an on-off switch provide the user interface. The Mica2 actually comes in three models, as detailed in the Crossbow MPR/MIB User Manual [14], all

transmitting at different frequencies. The models are the MPR400 operating at 915 MHz, the MPR410 operating at 433 MHz and the MPR420 operating at 315 MHz. All motes used in MoteLife were MPR400 models. All models use the CC1000 Very Low Power Transceiver. Chipcon, a company recently acquired by Texas Instruments, developed the CC1000. Like any modern multi-function wireless device, the power consumption varies according to the parts of the device that are in use at any time. If the processor is put into sleep mode it only consumes 8 μ A compared to 8mA when running at full power. The power consumption of the various parts of the Mica2 mote as given in the user manual [14] is shown in Table 1.

Device	State	Power Consumption
Processor	Full Operation	8 mA
	Sleep	8 μ A
Radio	Listening	411 μ A
	Receive	8 mA
	Transmit	5.3 to 26.7 mA
	Sleep	2 μ A
Log (Flash) Memory	Write	15 mA
	Read	4 mA
	Sleep	2 μ A
Sensor Board	Full Operation	5 mA
	Sleep	5 μ A

Table 1: Mica2 power consumption

In the MPR410 and MRP 420, the power level of the transmit signal can be set between -20 dBm and 10 dBm, giving varying power consumption from 5.3 to 26.7 mA as the output signal strength increases. The MPR400 (915 MHz) model, when transmitting at minimum strength of -20 dBm consumes 8.6 mA. The maximum output power of this model is 5 dBm consuming 25.4 mA. The various power levels and corresponding power consumption at 915 MHz are listed in Table 2, taken from the MPR/MIB User Manual [14].

Power Out (dBm)	Power Consumption mA
-20	8.6
-19	8.8
-18	9.0
-17	9.0
-16	9.1
-15	9.3
-14	9.3
-13	9.5
-12	9.7
-11	9.9
-10	10.1
-9	10.4
-8	10.6
-7	10.8
-6	11.1
-5	13.8
-4	14.5
-3	14.5
-2	15.1
-1	15.8
0	16.8
1	17.2
2	18.5
3	19.2
4	21.3
5	25.4

Table 2: Transmit (915 MHz) Power Consumption at various power levels

In order to calculate the power in use at any instant in time, one must therefore decide the status of all parts of the system and from that add up the total power consumption. To calculate the total power used over a given period of time, one must decide what portion of that time was spent by each part of the system in various states. These percentages are the duty cycles. Along with the duty cycle, one must also take account of the power level of the transmit portion of the radio cycle. From the figures given we can estimate the power consumption of a Mica2 mote, which is turned on, but has everything in sleep mode as shown in Table 3.

Part	Current in mAmps
Processor (Sleep)	0.008
Radio (Sleep)	0.002
Flash Memory (Sleep)	0.002
Sensor Board (Sleep)	0.005
Total	0.017

Table 3: Estimated power consumption in sleep mode

The estimated power consumption of a Mica2 that is constantly receiving and processing signals is shown in Table 4.

Part	Current in mA
Processor (100% on)	8.000
Radio (100% Receive)	8.000
Flash Memory (Sleep)	0.002
Sensor Board (Sleep)	0.005
Total	16.007

Table 4: Estimated power consumption receiving signals

The Mica2 is normally powered by two AA batteries wired in series to provide 3 volts, but other power sources can be used. The operational voltage range of the Mica2 is given in the user manual [14] as 2.7 to 3.6 Volts DC. Two AA batteries provide approximately 2000 mA hours at 3 volts. In order to quickly estimate the battery life of a Mica2 in sleep mode, we can divide 2000 mA hours by the current consumption of 0.017 mA, giving a result of 117,647 hours, approximately 4902 days, over 13 years. Doing the same calculation for a mote constantly receiving signals we find that the current consumption of 16mA will exhaust the batteries in just 125 hours, or 5.2 days.

Power management therefore is essential for obtaining reasonable lifetime for a mote. Running at full power, a Mica2 device can exhaust the power supply in less than a week, whereas most applications demand that they continue to run unaided for periods of six months or more.

Crossbow technologies produce many other motes, similar in size, specification and power consumption to the Mica2 mote. Their Mica2dot, no longer available, had the same specification as the Mica2, but only measured 2.5 cm in diameter. They now produce a postage stamp size version of one of their motes, the MICAz OEM module (Figure 2). As described in the datasheet [13] this mote has very similar power consumption to the Mica2. All the Crossbow motes run applications developed with TinyOS, an open source operating system for embedded systems, described in 2.2.



Figure 2: MICAz OEM Module

In 2002, a smart dust prototype called the spec (Figure 3) was built at UC Berkeley measuring just 5x5 mm. This was the first mote integrating radio communication and the TinyOS operating system on a chip this size.

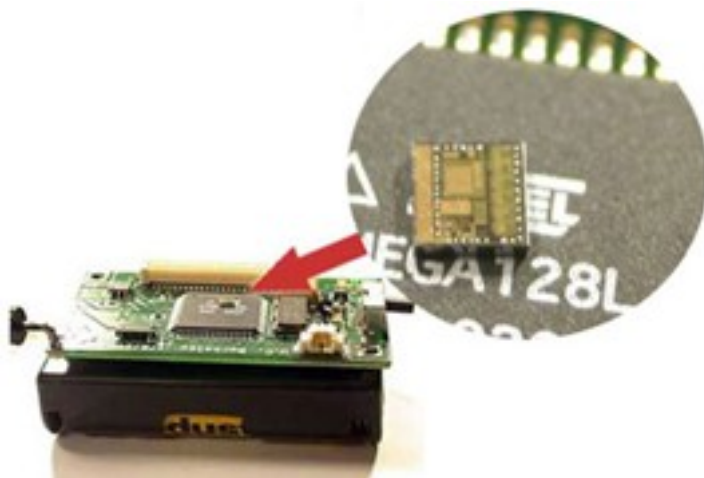


Figure 3: Spec mote on Mica2 mote

Intel Research in collaboration with the academic community is exploring the potential of motes in WSNs and commercial and industrial applications. The primary objective of their mote project is to produce an enhanced generation of motes. They have produced a prototype (Figure 4) measuring 3cm x 3cm, again using TinyOS as the operating system.



Figure 4: Intel Research prototype mote

In January 2007 Crossbow Technology announced the Imote2, a commercial release of the Intel Mote 2 design licensed from Intel. This mote integrates a high performance, low power, PXA271 Intel processor and an 802.15.4 radio with a built in antenna. It has 256 KB on-chip SRAM, 32 MB of SDRAM and 32 MB of Flash memory providing considerably greater storage capacity than earlier motes.



Figure 5: Imote2

In April 2007 Sun Microsystems released the SunSPOT (Small Programmable Object Technology, Figure 6), a mote built on a java virtual machine. The SunSPOT is similar in size to the Mote2 and is intended to ease WSN development. At the time of writing the device is not available outside of the United States. As part of this technology Sun developed the Squawk VM, a small Java virtual machine that runs directly on the processor. The SunSPOT allows WSN applications to be written using Java and an Integrated Development Environment (IDE). The SunSPOT is fitted with a 3.7 Volt rechargeable battery giving 750 mA hours, which is less than half of that supplied by AA batteries. The power consumption in standby mode is almost twice that of the Mica2 at 32 μ A. The combination of higher power consumption and lower battery capacity makes power management on the SunSPOT even more critical. Using both the CPU and radio will exhaust the battery in about 7 hours.



Figure 6: Sun Microsystems SunSPOT

Even though the technology is relatively young, the currently available motes are being used for many real world applications. Much effort is going into making the devices smaller and easier to program and making more efficient use of battery lifetime. No doubt many more applications will be developed for motes as they evolve.

2.2 Mote Development Environment

The vast majority of motes, including the Mica2 use TinyOS, an open source, modular, component-based operating system designed for concurrent embedded systems. TinyOS runs on a wide variety of motes and was originally created at UC Berkeley. A team there, in co-operation with Intel Research, still lead its development. Applications are written in nesC, a version of C tailored to the environment, with additional language features for concurrency and components. The component-based architecture requires having a clear bi-directional interface between components. Each module and component must state specifically what interfaces it uses. Each component must also define the interfaces that it provides. A component provides functionality to users through commands. Components also implement callback routines through events. A module using a component must provide all the events specified in the component interface. For example any module using the timer component must implement the fired event, which is triggered by the timer. A nesC application is composed of a number of modules and components wired together. Modules and components have configuration files, which detail how they are wired to other components. A nesC application has a top-level configuration file, which details how the modules and components inside are wired together. The TinyOS architecture allows different hardware specific components to be compiled for various platforms. Top-level code therefore remains independent of the underlying hardware, allowing applications to be easily ported to other motes. TinyOS Version 2.0 is a redesign and re-implementation of TinyOS written in 2006. This was due to requirements and uses that were not foreseen in the original implementation. While version 2.0 is not backwards compatible with version 1, code written for version 1 can be recoded for version 2.0. TinyOS also includes TOSSIM, a simulator for entire applications. The simulation is controlled via python scripts. There are also tools to create java code and classes directly from the nesC headers. These classes can then

be incorporated into a java application allowing it to communicate easily with motes. Once a nesC application has been developed it must be uploaded to the program memory. In the case of the Mica2, this is done via the programming board (mib510), available from Crossbow, which connects to the serial port (DB9) of the development computer. Computers without serial ports must use a DB9 serial to USB cable to connect. After attaching the mote to the programming board, a utility supplied with TinyOS is used to transfer the compiled code to the device.

2.3 Literature Review

Given the importance of power usage in relation to battery lifetime, it is not surprising to find that much research has taken place related to extending the life of motes and WSNs. Being new to this area, I found that there was a lot of material to cover.

At the hardware level, the design of power efficient circuits [4] has become more important thanks to the increasing popularity of mobile phones and Personal Digital Assistants (PDA). The interface between the hardware and the software can also affect power efficiency, through the provision of cross-layer enablers and power efficient modulation techniques [41]. In software, many techniques have been devised to preserve power at all levels of the Open Systems Interconnection (OSI) reference model [18], from the MAC layer right up to the application level. All aspects of the power saving mechanisms for a WSN must be integrated into a systematic software framework.

2.3.1 Avoiding Extra Transmissions

In order to save power, as seen earlier, the duty cycles of the various parts of a system must be kept in modes that use least power for as long a period as possible. Devices use the least power when they are not performing any action, known as the sleep mode. Radio transmissions in particular use a lot of power, so protocols favouring retransmissions are less suitable for WSNs. At an early stage, research showed that the classical multiple access protocols, Time Division Multiple Access (TDMA) and Aloha were not suitable for densely populated wireless networks without modification [8]. Pure Aloha has a large number of detected collisions and subsequent retransmissions, leading to large power expenditure for minimal data transfer as the number of nodes increases. Classical TDMA on the other hand has very good power consumption for the number of packets transmitted, but an extremely long delay as

the number of timeslots increases in proportion to the number of nodes on the network. As the 802.11 standards emerged for radio communication, the need for power conservation at the MAC layer continued to be of concern [52], particularly the need to minimise detected collisions and the need for retransmissions. Once collisions are avoided and the data is transmitted, error detection comes into play. Research showed that classic ARQ strategies, which have a large number of retransmissions, lead to a considerable waste of energy [59]. This triggered the development of adaptive ARQ schemes for radio, which reduce the data rate of the radio channel when it is impaired. This leads to less transmitted errors and thus less retransmissions due to errors.

Designing energy efficient radios for WSNs are considered in [47]. Specifically it focuses on energy efficient design techniques for the physical and MAC layers of the radio. Besides introducing power savings in individual blocks, such as the choice of modulation techniques for transmission, a designer can also trade off energy consumption across blocks.

B-MAC, a carrier sense media access (CSMA) protocol is introduced in [44]. By factoring out some functionality, such as channel arbitration and hidden terminal avoidance, and exposing control to a higher level, it provides a flexible interface to obtain ultra low-level power operation. It employs an adaptive preamble sampling scheme to reduce the duty cycle and reduce active listening.

The power efficiency of a MAC layer protocol can be measured as the energy expended in order to successfully transmit data [9]. Moving up from the MAC layer, we find that there are also opportunities to avoid sending extra transmissions at the higher levels of the stack. As a WSN sets up an ad hoc network, the choice of routing paths can have a significant effect on power consumption and many papers, discussed below, address this issue from various aspects. In order to reduce retransmissions due to errors, neighbours having the best quality transmissions should be selected for routing. Window Mean with Exponentially Weighted Moving Average (WMEWMA) [53] is a proposed method of estimating the quality of neighbouring links through snooping. Neighbouring nodes are then selected for routing according to link quality.

Many applications of WSNs involve collecting data at various locations, which must then be sent back to a common base station. This gives rise to the formation of a multi-hop network, where most of the nodes are required to pass information from

other nodes back up the line. If there is extra room in messages being passed on behalf of other nodes, a node may have the opportunity to piggyback its own data into the message, thus avoiding a future transmission. The maximum lifetime data-gathering problem is discussed in [31]. The aggregation or fusion of data shifts the focus from maximising node-to-node efficiency to a more data centric approach. The paper proposes a set of algorithms to maximise the lifetime of a data collection network for cases where nodes are permitted or not permitted to aggregate data. Many WSNs are designed to sense data and then forward it to a base station. Another approach to reducing the number of transmissions required, explored in [20], is to store the data in the WSN using data aggregation, drill-down query processing and graceful data ageing. A form of structured query language (SQL) for mote networks has also been proposed [38].

2.3.2 Reducing Radio Cycles

In order to reduce the duty cycle of the radio a number of mechanisms can be used including letting the radio sleep until it is needed, rather than have it constantly listen for signals.

If a radio is switched on in a node, it will consume extra power when a signal is heard, as reception will be activated. In a densely populated WSN, this may lead to many nodes wasting energy receiving signals that are not meant for them. In order to avoid this, Power Aware Multi-Access protocol with Signalling (PAMAS) was introduced [48]. In this protocol a node will switch off the radio if a signal is overheard that is being received by another node. Control signalling for request to send and clear to send is carried on another channel not used for transmission.

A Basic Energy Conserving Algorithm (BECA) was introduced [57], which leaves the radio turned off for much of the time. Using expected ratios of on/receive/off in densely populated WSNs, spare node capacity can be used to forward messages. A node needing to send a signal of course will turn on the radio. A companion algorithm, Adaptive Fidelity Energy Conserving Algorithm (AFECA) uses observations on the node density to modify the ratios.

Another system that makes use of spare capacity in densely populated networks is Span [6]. This is a randomised distributed algorithm, where nodes decide locally whether to participate in message forwarding, based on an estimate of the number of neighbours that would benefit from them staying awake together with the residual

power in their battery. Span runs above the MAC and link layers and interacts with the routing protocol.

Similarly, Geographical Adaptive Fidelity (GAF) [58] is an energy conserving ad hoc routing protocol, which uses geographical information to turn off superfluous nodes. The area of the WSN is divided into a grid, where all nodes within a grid are equivalent from a packet passing perspective. Nodes within a grid decide between themselves who sleeps and for how long. Nodes periodically wake up and trade places to achieve load balancing.

The systems discussed so far attempt to let radios sleep, based on assumptions about traffic patterns and redundancy due to dense node populations. What is really needed is a method of letting the radio sleep until a signal is detected and then waking the radio. In some systems, typically for telephone traffic, the radio transmissions use a separate channel from the control signals [46]. This allows the radio to sleep for most of the time, while a low power receiver monitors the control channel for incoming signals. When one is detected a wakeup message is sent to the main radio for signal reception.

As most nodes do not have the capacity to use a separate control channel, schemes can be devised where nodes wake up periodically and listen [26]. Sentry nodes notify other nodes of the event and nodes do not enter sleep mode while signalling is taking place. By careful selection of the scheduling periods some power saving can be achieved, however this can be hard to configure for some applications.

Another wake up method suitable for WSNs is to use a special hardware component, a radio triggered circuit, to wake up the node when the signal transmission of another node is detected [22]. Such a circuit does not use any power, but is powered by radio waves, which supply enough current to trigger the wakeup.

TDMA has the potential to avoid collisions entirely in WSNs. However as previously discussed, in densely populated networks it is not feasible to allocate one time slot to each node as the latency introduced becomes unacceptable. TDMA also introduces the problem of synchronisation. The nodes must agree on when the frame boundaries occur, so that they are in agreement on when each node may broadcast. A WSN has extra complications for TDMA introduced by unidirectional links, unreliable links and sleeping nodes. A solution for using TDMA, which addresses these problems, is proposed in [33]. Following stabilisation or diffusion of the network no contention occurs as the time division multiplexing is operational.

In [32] the TDMA algorithms are developed to allow customisation of timeslot allocation for three communication patterns: broadcast, converge cast and local gossip. Rectangular and hexagonal grids are used to evaluate the algorithms.

TDMA for the most common type of application, data gathering, is addressed in [25] with the proposed Flexible Power Scheduling (FPS) protocol. This protocol aims to reduce power and support fluctuating demand in the network without any central control. The energy savings come from powering down nodes, identified through dynamic scheduling, during idle periods. It features a two-tier architecture, for coarse grain scheduling to plan radio on/off times and fine grain scheduling for channel access.

Data gathering and TDMA channel assignment is further examined in [19]. This paper proposes algorithms to determine the smallest length conflict free assignment of slots during which packets generated at nodes reach their destination. Spatial reuse of timeslots is used in multi-hop WSNs where nodes are in non-conflicting parts of the network. After discussing two centralised algorithms, they are compared with a token-based distributed algorithm. The distributed algorithm, not having an overall knowledge of the network topography, does not perform as well as the centralised versions and is an area for future work in this area.

2.3.3 Power aware routing

Traditionally routes were selected according to shortest path. In a WSN, the power constraints imposed make selection of route in relation to power consumption more important. Singh et al. [49] discussed the need to make routing power-aware and proposed 5 metrics for power-aware routing. These are:

Minimise the energy used per packet

This is intuitively the way to ensure that minimum power is used to transmit a packet. However, in situations where one node connects with many others, this metric alone will result in multi-connected nodes being overused and suffering failure before other nodes.

1) Maximise the time to network partition

If there are few nodes connecting separate parts of the network, it is important to keep them alive as long as possible so that the network does not partition. Optimising this metric while maintaining high throughput and low delay is difficult.

2) Minimise variance in node power levels

This metric tries to ensure that all nodes remain up and running together for as long as possible.

3) Minimise cost per packet

Paths selected should not contain nodes with battery reserves significantly lower than the others. The cost per packet is a reflection of a nodes unwillingness to participate, which increases as reserves are depleted. This metric measures the total unwillingness of all nodes in the selected route.

4) Minimise maximum node cost

Even if the cost per packet is minimised, there may be nodes on the path with a very high cost. This metric ensures that individual nodes along the path are protected.

Local Energy Aware Routing (LEAR) [54] builds upon these ideas to provide a distributed power-aware routing algorithm, based on Dynamic Source Routing (DSR). In this scheme local nodes decide whether or not to participate in routing based on their remaining battery level. A depleted node can preserve remaining power, by refusing to forward packets. If the power level is below a threshold, nodes refuse to participate. This can lead to network partitioning if a route cannot be found to the destination. To avoid partitioning the network too early, the threshold is lowered when a retransmission is detected. The protocol is non-blocking as the destination node can respond immediately on reception of a signal.

Online Power Aware Routing (OPAR) [36] introduces the *max-min* zP_{min} approximation algorithm, which tries to strike a balance between the route with minimum power consumption and maximising remaining power. The Dijkstra algorithm [15, 11] is used to find the route with the shortest path in terms of power consumed, but taking into account the remaining residual power. A parameter z is used to balance the optimisation, controlling the trade-off between the power consumed and the residual power. A random value is initially selected for z and this is refined until best results are obtained. This method has a high overhead, especially in large networks, so a zone based hierarchy is proposed to counteract this. The network is divided into zones and optimisation is local to each zone.

Power Aware Source Routing (PSR) [39] is a power balancing protocol. The cost of each transmission is calculated at each node with regard to transmit power and residual power. While routes are being established, a node listens for some time to all request signals, which are supplemented with the cost calculation of the sending nodes. When the timer expires the node will respond with the lowest cost route. This introduces some latency in route establishment, which is justified by the advantage of network power balancing.

Many power efficient routing protocols have been proposed for WSNs including Hopping [3] a bi-directional stigmergy based protocol. This works in a similar manner to that used by ants laying down pheromones to communicate the best routes.

2.3.4 Adjusting Transmit Power

As nodes can transmit at a number of different power levels, much research has examined the opportunity for reducing power consumption by adjusting the transmit power level to suit the environment. Gupta and Kumar determined the critical power a node should transmit at in order to ensure that all nodes are connected [23].

Adjusting the power to this level ensures that interference with neighbours is minimised and power consumption is kept low.

Minimum energy routing for wireless mobile networks is proposed in [45] using GPS receivers in each node to help with routing. A distributed algorithm is used to find the minimum power topology in each node. Simulations show that the average power consumption per node is considerably low.

In [5] a study was taken to consider routing mechanisms, which maximise the lifetime of the network when the transmission power can be varied to increase or decrease the number of neighbours. The results showed that maximum network lifetime is achieved when the energy used for signalling was balanced in proportion to residual energy.

Adjustment of power levels for connection-oriented communication is considered in [40]. A set of heuristics was developed to enable nodes to determine end-to-end paths with sufficient resources and appropriate transmission levels. As seen earlier low power consumption alone is not enough to delay network partitioning and node over utilisation. A power cost metric combining node lifetime and transmission power was proposed in [50].

PARO, a Power Aware Routing Optimisation, is described in [21] where all nodes are located within the maximum transmission range of each other. Nodes listen to transmissions and calculate the minimum power to reach the node using a propagation model. If an intermediate node determines that it can forward a packet using less power, it elects to become a redirector. Nodes elect to be redirectors on behalf of a source-destination pair, in order to reduce overall power consumption. The idea is that the overall power consumption can be minimised using an intermediate node as the power level needed for the node pair to communicate directly would be greater. To cope with mobility a route maintenance algorithm is added.

Minimum Power Configuration Protocol (MPCP) is proposed in [55]. This integrates topology control, power aware routing and sleep management into a protocol that can dynamically configure a network to minimise energy consumption. When the network load is light, most nodes are in sleep mode, so transmissions should use larger hops to minimise the number of nodes that need to be woken up. On the other hand when network traffic is heavy, short hops with minimum transmission power should be selected to minimise power.

An empirical evaluation of adaptive transmission protocols is undertaken in [27]. The effects of implementing Dynamic Transmission Power Control (DTPC) are examined compared to Fixed Transmission Power Control (FTPC) using a large Mica2dot test bed. For traffic patterns that diverge from a common source, FTPC is sufficient to achieve the maximum throughput. For traffic converging to a common sink, DTPC achieves a 16% drop in power consumption while not affecting throughput. When aggregation is used, DTPC does not offer any advantage over FTPC.

Following studies confirming that communication quality varies with time and environment in wireless networks, Adaptive Transmission Power Control (ATPC) was proposed [37], in which each node builds a model for every neighbour monitoring transmission power and link quality. A feedback based transmission power control algorithm is then used to dynamically maintain individual link quality through signal strength adjustment. The protocol is found to be robust while delivering power saving.

3 Work Undertaken

A Mote has a finite life, largely limited by two factors, the power supply and the flash memory. The most common source of power in motes is a battery. Each operation performed depletes the battery, until there is no power left and the mote dies. A mote that dies in this manner can be brought back to life by replacing the battery, so it is more akin to being in a coma than dying. Whether or not it is feasible to replace the battery depends on the application and environment of the mote. In military applications it is likely to be cheaper to drop more motes into the network than to replace the batteries. Motes use flash memory for long term memory. This memory is used to retain data while the device has no power or is turned off. A mote can record measurements to the flash memory until the power dies. If the battery is replaced, the recorded measurements can then be retrieved.

The number of times that a flash memory can be rewritten is finite, usually in excess of 100,000 operations. Once this limit has been exceeded, the memory becomes unusable, thus rendering the mote useless for many applications. Unlike the battery, it is not cost effective to replace flash memory. A scheme can be implemented to ensure that flash memory is used in a round robin fashion so that the start of the memory is not used more than any other part, as normally happens in logging applications. It is also possible to reserve a section of the flash memory to record the number of write operations and therefore estimate the remaining life of the flash memory. However, writing to this area repeatedly can overuse this section of memory thus defeating the purpose of the scheme. Such schemes need standardisation at the operating system level as the possibility of any application bypassing the scheme renders it useless. In practice, the lifetime of the flash memory for most applications is much greater than the number of measurements recorded.

This thesis focuses on power consumption. The results of measuring power drain on Mica2 devices when constantly computing, constantly transmitting, constantly receiving and constantly sensing are recorded. A method for providing individual nodes with self-awareness of their life expectancy is developed through the addition of a self-preservation module. A node aware of its impending demise can take evasive action to preserve power. For example, a node having collected imported data could enter a low power listening mode and refuse to process any signals until it

is asked to return collected data. A rule-based decision making scheme is described which eases the implementation of such applications.

3.1 Battery Drain

The amount of time a battery will last in a mote can be estimated by dividing the milli Amp hour (mAh) rating by the current drawn. Unfortunately this is only an estimate and the actual lifetime is affected by the discharge voltage, discharge profile, temperature and humidity. The discharge voltage varies over time. As the voltage affects the current drawn, this will also vary during the life of the batteries. The discharge profile also has a major effect on battery lifetime. Firstly drawing currents that are large in proportion to the mAh rating of a battery can decrease the efficiency of the battery. This is known as the capacity offset. The current drawn should be a fraction of the mAh rating for best results. A discharge profile containing periods of minimal power consumption also increases battery life. Peukert's equation [42] can be used to approximate how the available capacity of a battery changes in response to the rate of discharge. This formula is

$$C = I^n T$$

Where C is the theoretical capacity of the battery, I is the current drawn, T is time and n is the Peukert number, a constant for any given battery directly related to the internal resistance of the battery. This number is determined empirically.

Battery type also affects the discharge properties. Some batteries such as Lithium Ion have a fairly flat discharge curve and provide an almost constant voltage until near the end. Others such as Lead acid have a voltage, which reduces over time. At low temperatures batteries can freeze and stop functioning. At the other extreme, high temperatures may cause chemicals to break down and destroy the battery. Within the operating temperature, batteries usually perform better in higher temperatures. While being stored, warmer temperatures increase chemical reactions and reduce battery life. Mindful of the number of variables affecting discharge, the characteristics of a Mica2 mote powered by a pair of AA batteries wired in series to provide the operating voltage are examined. The MPR/MIB user manual for the Mica2 [14] shows that the power consumption of the radio, operating in the 902-928 MHz band, when

transmitting ranges from 8.6mA to 25.4mA depending on the power setting. When receiving the power consumption is 10mA and less than 1 μ A while sleeping. The user manual gives power consumption for the CPU as 8mA when active and 8 μ A while sleeping. The manual also gives figures for the flash memory operations as 15mA while writing, 4mA while reading and 2 μ A while sleeping. The sensor power consumption is given as 5mA for sensing and 5 μ A while sleeping. In order to calculate the power consumption per hour it is therefore necessary to establish the duty cycles of the radio, CPU, sensors and flash memory. For example, in order to discharge the battery very quickly you could run the CPU and sensor at 100%, while continuously transmitting at full power on the radio. This would give a constant current drain of 8mA + 5mA + 25.4mA = 38.4mA. The 5 μ A for the sleeping sensor and 2 μ A for the sleeping sensor bring this to 38.407mA and therefore have a negligible effect. In most normal applications all devices would be sleeping for 99% of the time giving a current drain of less than 1mA.

Turning to the batteries, a typical AA battery is rated at about 2000 mAh for typical mote currents at 3V. This rating divided by the power consumed gives us the expected hours of battery life. So to calculate how long our previous example of constantly transmitting at maximum power would take to deplete the battery divide 2000 mAh by 38.4 mA which gives us just over 52 hours of battery life. In a more realistic application, the current is more likely to be in the region of 1mA or less. 1 mA would give us 2000 hours or just over 83 days in ideal circumstances.

The Mica2 device has an accurate voltage reference that can be used to measure battery voltage. The voltage is not obtained from the reading directly, but from the following formula taken from the MPR/MIB user manual for the Mica2 [14]:

$$V_{batt} = V_{ref} \text{ADC_FS} / \text{ADC_Count}$$

Where V_{batt} is the battery voltage, $V_{ref} = 1.223$, $\text{ADC_FS} = 1024$ and ADC_Count is the measurement made. A measurement of about 419 therefore indicates a full charge of 3V, while a value in excess of 522 indicates that the battery voltage is less than 80%. A comparison with a voltmeter confirmed that the motes were returning accurate measurements.

3.2 Measurements of Battery Drain

In a series of experiments the discharge of the battery over time was recorded while the Mica2 device performed various activities. The idea was to utilize just one part of the mote at 100% of its cycle, while keeping the remainder in sleep mode. This allows measurement of battery drain for the operation of different parts of the system. The expected duration of the experiment was calculated, using the method described earlier. Two new alkaline AA batteries were attached to the mote at the start of each experiment. Only two types of battery were used in the experiments, Duracell Long Life and Philips Powerlife. Each experiment was carried out at least twice and there was no measurable difference between these two battery types.

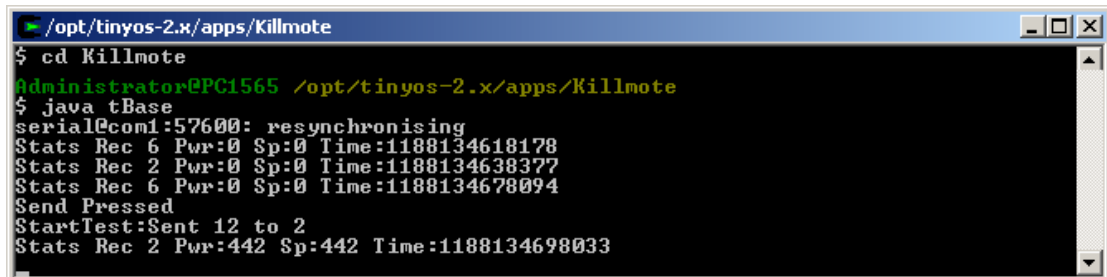
The method of recording was constant across all experiments. A TinyOS application, KillMote, was written and loaded on all motes. The KillMote application communicates with a modified BaseStation application that comes as part of TinyOS. A program was devised to send instructions to the mote via a graphic user interface. Each mote can then be instructed to perform a particular experiment. By necessity the measuring itself consumed power. A one second timer was used to count minutes. At the end of each minute a recording of the voltage reference on board the Mica2 was made and recorded in Flash memory. One LED was also flashed briefly during the measuring time at the end of each minute, so it was possible to tell whether a mote had died or not by waiting a minute for the LED to flash. A No Operation experiment was performed to monitor the power consumption of the measuring itself.

The MPR/MIB user manual [14] notes that ADC channel 7, which is used to measure the voltage, is also used for JTAG debugging on the Atmega128 processor. The Mica2 ships with the JTAG fuse enabled which lowers the impedance of channel 7 and affects the voltage reference measurement. This fuse was disabled for all experiments to ensure accurate results.

When the green LED was observed for 2 minutes and failed to flash the experiment was deemed to be over. The mote was turned off, the batteries replaced with working batteries. A routine was then run to extract the data stored in the Flash memory. The tests performed were, Continuous Signal Transmission, Continuous Signal Reception, LEDs Always On, Continuous CPU Activity and Continuous Sensing.

3.2.1 KillMote User Interface

TinyOS is supported on two platforms, Cygwin (a Linux like environment for Windows) and Linux. Cygwin allows you to recompile and run Linux applications under Windows and was the platform used for this project.



```
Administrator@PC1565 /opt/tinyos-2.x/apps/Killmote
$ cd Killmote
Administrator@PC1565 /opt/tinyos-2.x/apps/Killmote
$ java tBase
serial@com1:57600: resynchronising
Stats Rec 6 Pwr:0 Sp:0 Time:1188134618178
Stats Rec 2 Pwr:0 Sp:0 Time:1188134638377
Stats Rec 6 Pwr:0 Sp:0 Time:1188134678094
Send Pressed
StartTest:Sent 12 to 2
Stats Rec 2 Pwr:442 Sp:442 Time:1188134698033
```

Figure 7: Cygwin running in Windows

For the experiments a Java program, called tBase, was written to communicate with the motes through the BaseStation mote. As each mote running the KillMote application was switched on, it broadcast a signal to the base station and turned on the green LED, to give a visual indication that it was ready. When tBase detected a signal from a new mote, the number of the mote was added to the list of motes on the left side of the screen. The last data received from a mote was also displayed at the top of the screen. Figure 8 shows tBase after detecting motes 6 and 2. The line at the top of the screen shows that the last signal received was a Stats signal from mote 2, indicating an ADC channel 7 reading of 442.

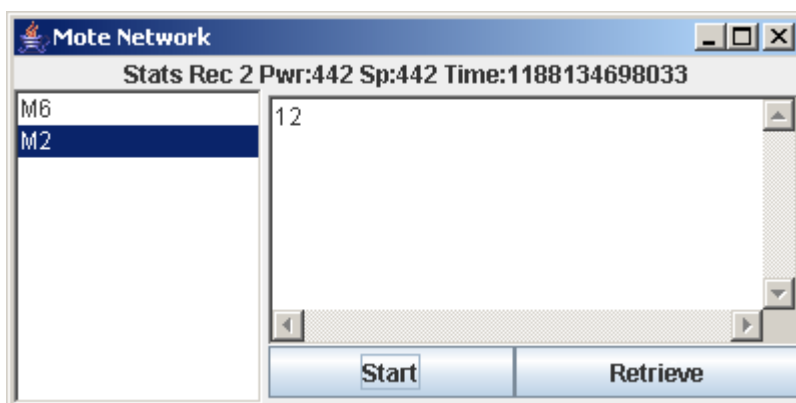


Figure 8: tBase Java application

In order to send a message to a mote the user first selects the mote from the list on the left of the screen. Any number entered into the area on the right of the screen is then sent to the mote when the user clicks on the Start button. In Figure 8 mote 2 has been selected, so clicking on the Start button sends the number 12 to mote 2. The numbers sent to the KillMote application result in the actions listed in Figure 9.

Number	Action
1	Erase the Flash Memory
2	Start Continuous CPU Activity Experiment
3	Start Continuous Signal Transmission Experiment
4	Start Continuous Signal Reception Experiment
5	Start Continuous Sensing Experiment
6	Get Data From Flash Memory
7	Stop test and close Flash Memory
8	Get remaining space in Flash Memory
9	Start Continuous LEDs on Experiment
10	Start No Operation Experiment
11	Transmit Continuously for Reception Test
12	Read Voltage Sensor
13	Read Light Sensor

Figure 9: KillMote commands

On reception of a signal, the KillMote application extinguished the green LED, which as mentioned was flashed briefly during the experiments. When the experiment ended the batteries were replaced and the mote was rebooted. The tBase program was then used again to retrieve the stored data from the Flash memory. When the Retrieve button is clicked, the tBase program continues to query the mote until all data have been gathered and verified. The results are then written to a file on the hard disk. The experiments carried out are detailed below.

3.2.2 Experiment 1: No Operation

In order to see what effect the measuring process itself had on the mote, a no-operation experiment was carried out, where the mote performed no operation, other than make voltage measurements and record the results. For the no-operation experiment, the radio was turned off and no sensor board was fitted. The LED was only used for a brief period each minute, while the CPU was used briefly each second and more intensively at the end of each minute. The calculation of the expected power consumption and life expectancy is shown in Table 5.

Part	Active	Current in mA
Processor	1.67%	0.141
Radio (Sleep)	0.00%	0.002
Flash Memory	0.21%	0.033
Sensor	0.42%	0.026
<hr/>		
Total		0.202
	Hours	9855
	Days	410.63
	Months	13.69

Table 5: No Operation Experiment estimated duration

The CPU has to wake up for a very short period each second to increment a counter and is then active for slightly longer at the end of each minute while a sample of the voltage is taken and the result is written to the flash memory. An estimate of the CPU activity was based on the assumption that it should be awake for a total of about 1 second in each minute, giving a total of 1.67%. The sensor and flash memory are used for a short period each minute, so the sensor time was estimated as 250ms each minute or 0.42%. Data to be written to the flash are cached until the data received reaches a certain size. Assuming actual writes occur 50% of the time, a figure of 125ms was used for the estimation of the flash cycle, giving 0.21%. The estimated current consumption of 202 μA gives an expected battery lifetime in excess of a year. Unfortunately, the time available for this project was not sufficient to let this experiment run to completion, so the mote had to be turned off after operating for 660.4 hours or 27.52 days.

The voltage drop for this experiment is shown in Figure 10. Even though the mote had to be turned off before the battery was depleted, the voltage had dropped below 2.5 Volts. This is outside the operational voltage given in the user manual [14] indicating that the current consumption was far greater than the expected 202 μA .

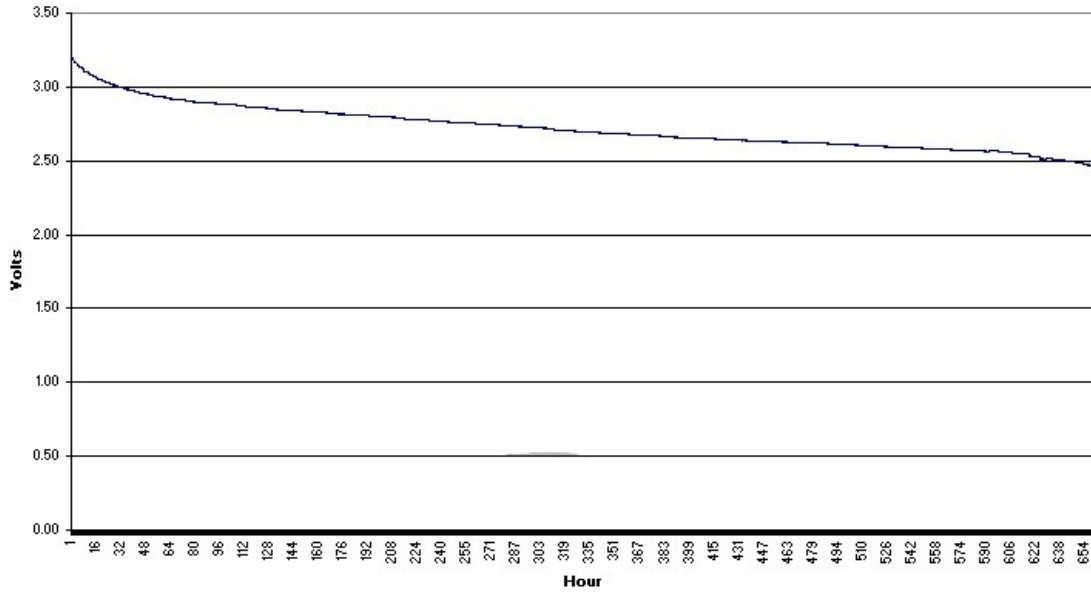


Figure 10: Exp A, Voltage Drop No Operation

The current for this experiment when measured with an amp meter is 2.92 mA, giving an expected lifetime of 684.93 hours or 28.54 days. This, together with the rapid depletion of the battery, indicates that the duty cycles were underestimated.

3.2.3 Experiment 2: Continuous Signal Transmission

For the continuous transmission experiment, no sensor board was fitted. The mote was set to transmit at 0 dBm, which is the default. The mote was programmed to transmit a signal when instructed to start this experiment. Sending a signal from the Mica2 is a split phase operation. Control is passed to the operating system to send the signal and an event is triggered when the signal has been sent. During this experiment this event was used to transmit another signal, thus starting cyclic signal sending. As the radio is active during this experiment, any signals overheard by the mote would put it into receive mode. It was important therefore to ensure that the mote was isolated from any other transmitting motes. While performing this experiment motes were placed in metal filing cabinets isolated from other transmitting motes, to prevent overhearing signals. A sniffer mote, which toggles a LED when a signal is heard, was used to confirm that the environment around and inside the filing cabinet was free from other signals. Once the experiment started the sniffer mote was used to confirm that the mote was indeed transmitting. The calculation of the expected power consumption and life expectancy is shown in Table 6.

Part	Active	Current in mA
Processor	100%	8.000
Radio (Transmit)	100%	16.800
Flash Memory	0.21%	0.033
Sensor	0.42%	0.026
Total		24.859
	Hours	80.45
	Days	3.35
	Months	0.11

Table 6: Constant Transmission estimated duration

The CPU is expected to be in use throughout the process, so it is 100% active for this calculation, giving a current of 8 mA. One would also expect the radio to spend 100% of its time in transmit mode, which at 0dBm consumes 16.8 mA. This along with the small current used for measuring the voltage every minute, gives an estimated power consumption of 24.86 mA, which should deplete the battery in just 3.35 days or 80 hours.

In 2 runs of this experiment, B and C, the motes ran for 101.6 hours and 118.5 hours respectively. This is 26% and 47% greater than predicted. Crossbow quotes the operating voltage of the Mica2 as 3.6 V to 2.7 V [14]. However the motes continued to operate down to 2.42 V and 2.39 V. The voltage drop over time for experiment B is shown in Figure 11 while the voltage drop for experiment C is shown in Figure 12.

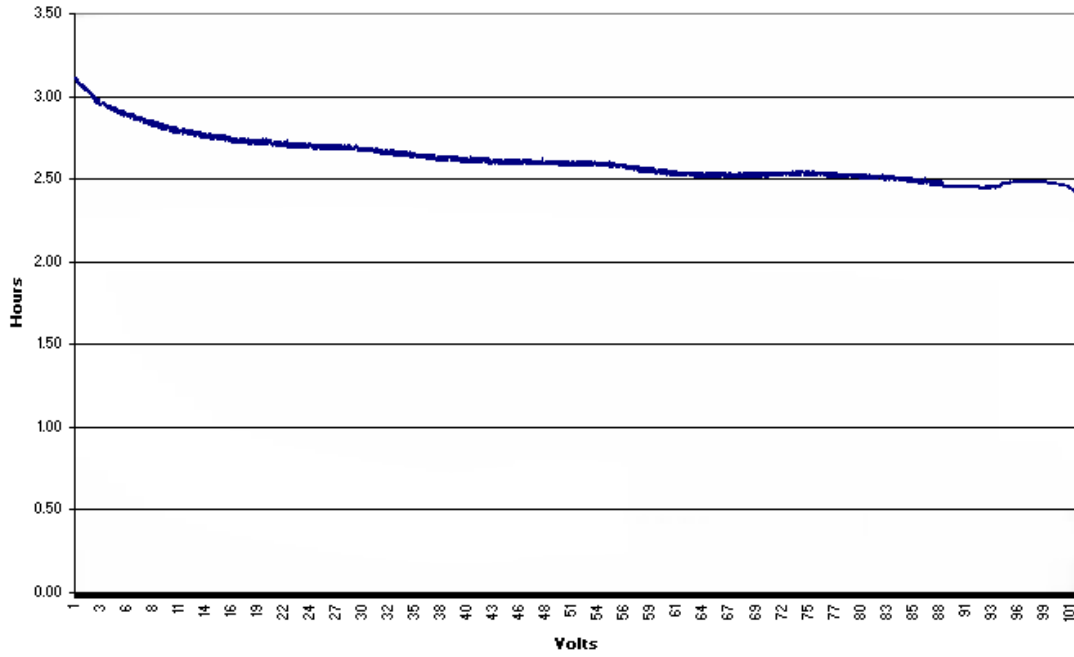


Figure 11: Exp B, Voltage Drop for constant transmission at 0 dBm

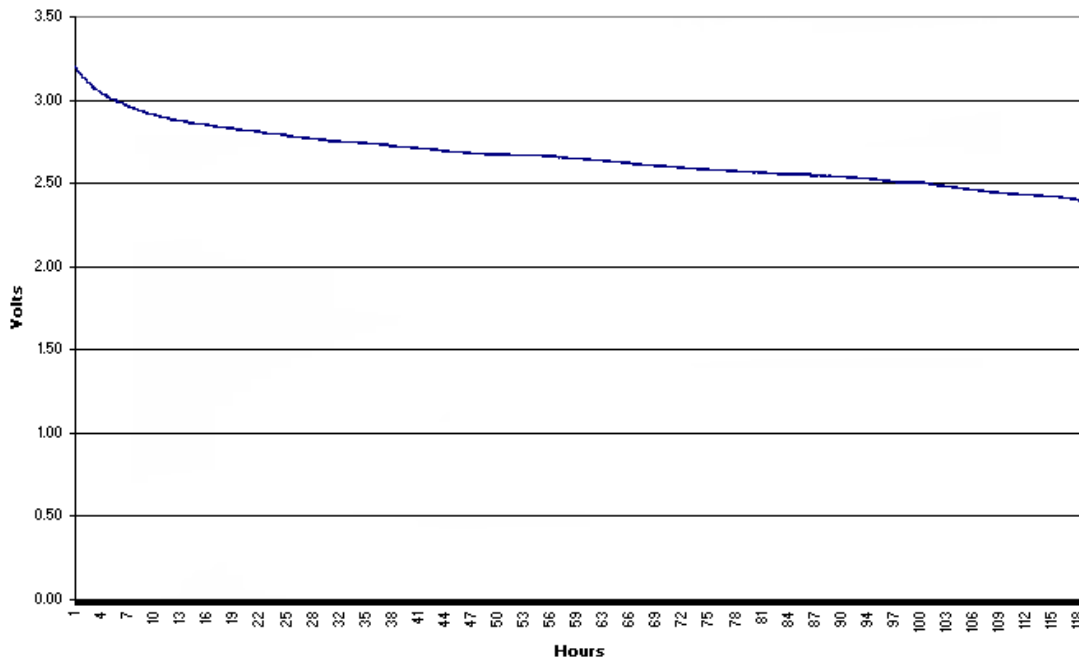


Figure 12: Exp C, Voltage Drop for constant transmission at 0 dBm

The motes first touched the bottom of the operating voltage range, 2.7 Volts in 28 and 41 hours respectively, just 35% and 51% of the predicted lifetime. In both cases 2.7 Volts was reached in less than half of the actual lifetime of the mote.

The current for this experiment when measured with an amp meter is 16.94 mA, giving an expected lifetime of 118.06 hours or 4.92 days. Experiment C ran for almost that exact number of hours, while experiment B finished 17 hours earlier.

3.2.4 Experiment 3: Continuous Signal Reception

For the continuous reception of signals, a pair of motes were used, one to transmit constantly and the other to receive. A modified version of the transmit mote was used which starts transmitting on startup and does not record measurements. Once the experiment was started on the receive mote, the transmit mote was started. During this experiment the transmit mote was kept on the programming board so that it was running on mains power and the mote could continue to transmit until the batteries on the receiving mote had been depleted. A sniffer mote was also used in this experiment to ensure that the transmitting mote was indeed transmitting. The calculation of the expected power consumption and life expectancy is shown in Table 7.

Part	Active	Current in mA
Processor	100%	8.000
Radio (Receive)	100%	8.000
Flash Memory	0.21%	0.033
Sensor	0.42%	0.026
Total		16.059
	Hours	124.54
	Days	5.19
	Months	0.17

Table 7: Continuous Signal Reception estimated duration

In the receiving mote, the radio should be receiving signals for 100% of the time. As a result the CPU should also be active for 100% of the time. This together with the current used for measuring gives an estimated power consumption of 16.059 mA giving an estimated lifetime of 5.19 days or 125 hours.

In 2 runs of this experiment, D and E, the motes ran for 117.5 and 130.2 hours. For experiment D this was 5.7% less than predicted and for experiment E it was 4.5% longer. This was the most accurate prediction of all the experiments. The voltage drop over time is shown in Figure 13 for experiment D and Figure 14 for experiment E.

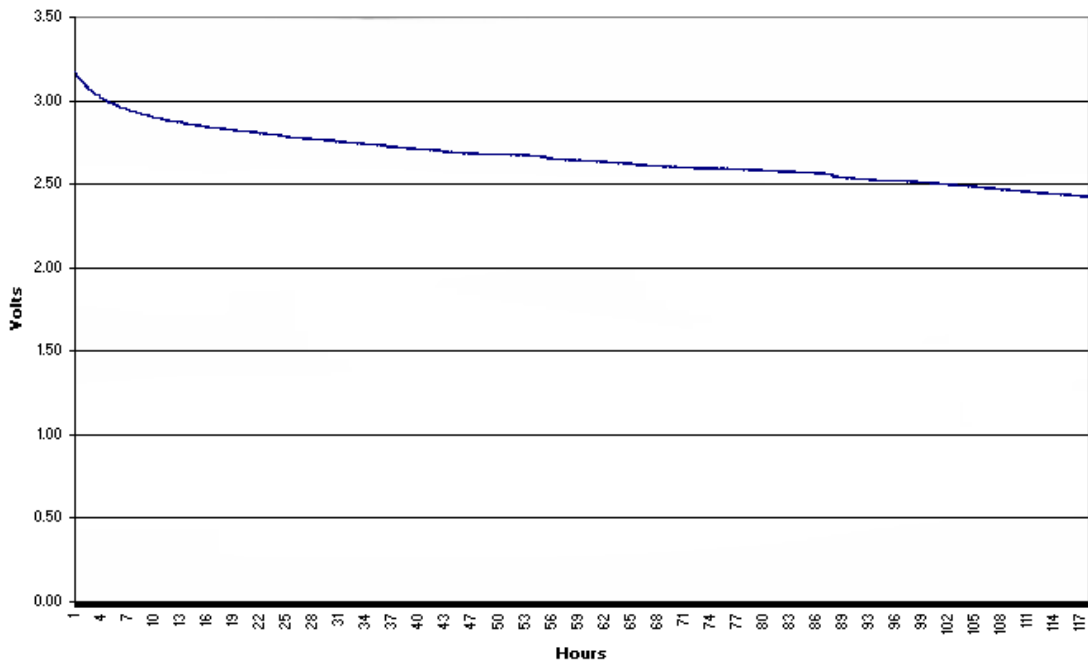


Figure 13: Exp D, Voltage Drop Constant Signal Reception

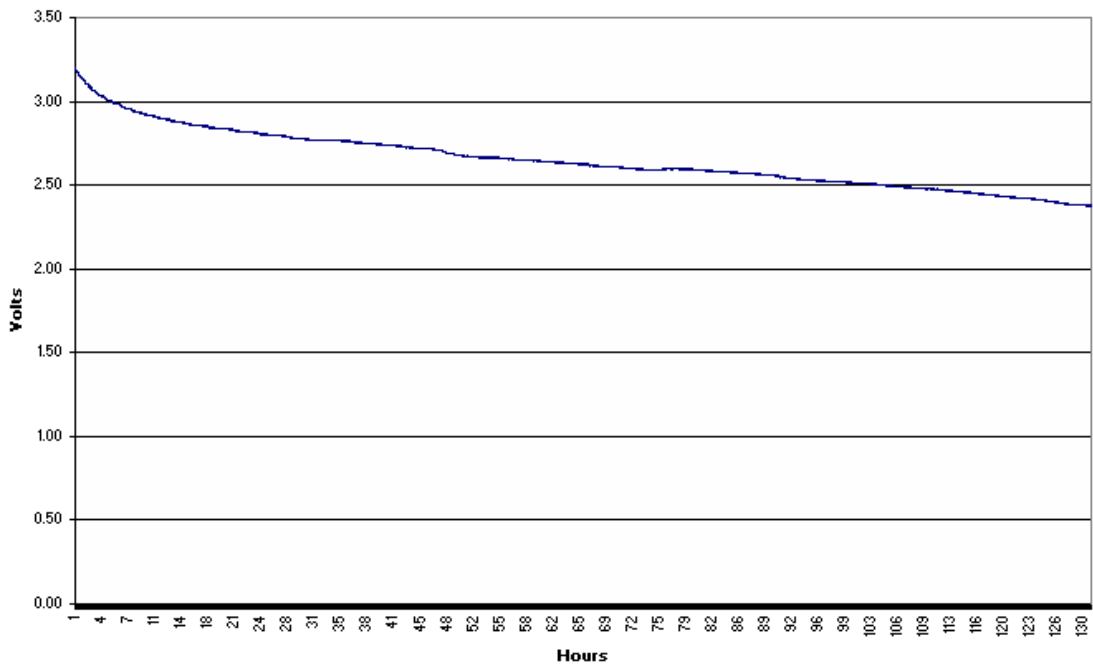


Figure 14: Exp E, Voltage Drop Constant Signal Reception

Again we can see that the motes continued far beyond the stated operating voltage, down to 2.43 V for experiment D and 2.38 V for experiment E. In both experiments 2.7 V was reached well before the end of operations.

The current for this experiment when measured with an amp meter is 15.75 mA, giving an expected lifetime of 126.98 hours or 5.29 days. Experiment D finished almost 10 hours shorter than this, while experiment E lasted 3 hours longer.

3.2.5 Experiment 4: LEDs On

This experiment was not on the list of scheduled experiments originally, as the relevant literature does not make any reference to the power consumption of LEDs. The power consumption of a single LED is typically 15 to 20 mA, but can vary according to manufacturer, colour and type of LED. One would assume that low power devices are fitted to motes, but such figures prompt examination. The LEDs on experiment was devised to simply turn on the three LEDs and turn everything else, except the measuring, off. The radio was turned off and no sensor board was fitted. As the power consumption calculation does not take the LEDs into account, the current should be identical to the no operation experiment. Power consumption should be 202 μ A and the battery should last for more than a year. Rather than blink the green LED on briefly at the end of each minute, the program was set to blink the LED off at the end of each minute during this experiment. A trial run of this experiment over four days, using somewhat depleted batteries, showed a voltage drop from 2.77 volts to 2.47. A 202 μ A current is not consistent with this result and the batteries would certainly not last a year, so the experiment was scheduled. This gave a problem with how to estimate the duration of the experiment. Using the voltage drop of the trial run, a drop of 0.3 V in 4 days, or a 0.075 V drop per day, was observed. Running the experiments showed that a fully charged battery starts at 3.2 V and operates down to approximately 2.5 V giving a total voltage drop of 0.7 V for the life of the battery. If the loss every day were 0.075 V, a full battery would be depleted in roughly 10 days or 240 hours. Given that a full battery gives 2000 mA hours, dividing by 240 hours shows that a current draw of 8.3 mA is required. Assigning 0.3 mA to the measuring mechanism, we get a rough estimate for this experiment. The calculation of the expected power consumption and life expectancy is shown in Table 8.

Part	Active	Current in mA
Processor	1.67%	0.141
Radio (Sleep)	0.00%	0.002
LEDs (3 On)	100%	8.000
Flash Memory	0.21%	0.033
Sensor	0.42%	0.026
Total		8.202

Hours 243.84
Days 10.16
Months 0.34

Table 8: LEDs on estimated duration

Due to time constraints and the late discovery of the scale of LED power consumption, only one LED experiment was carried out. Experiment F took 301.6 hours to deplete the batteries, 23.7% longer than the rough calculation predicted. The voltage drop over almost 13 days is shown in Figure 15.

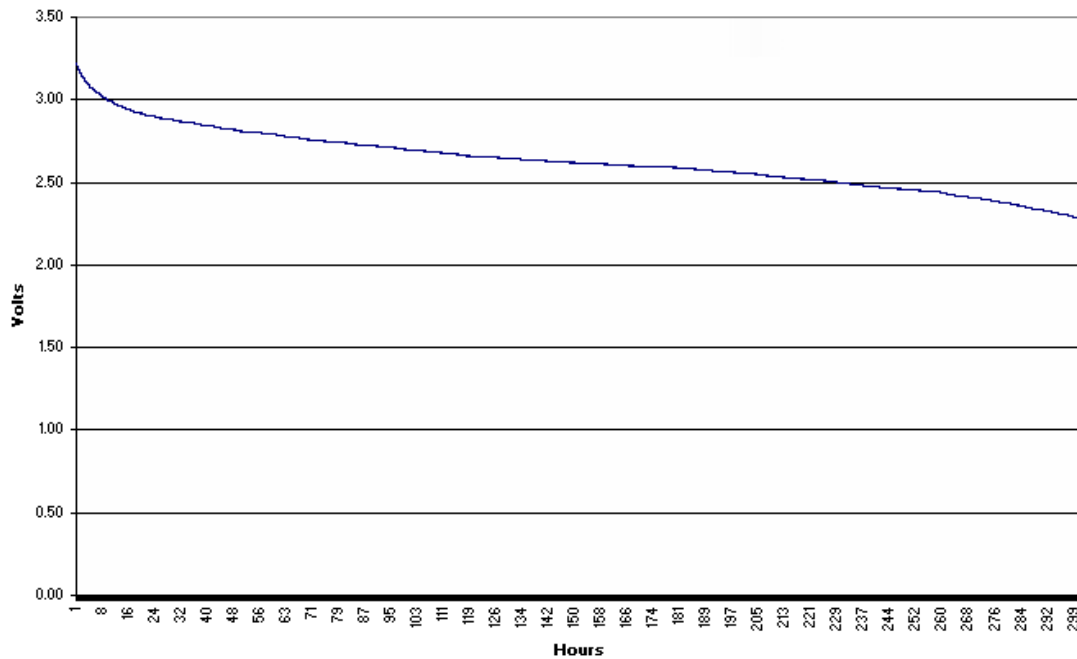


Figure 15: Exp F, Voltage Drop LEDs on

The mote here continued to operate right down to 2.28 Volts. On the last day the green LED went out many hours before the other two. This could be due to the mote not having sufficient power to blink it at the end of each minute.

As the calculation was very rough for this experiment, it is advantageous to consider what power consumption the LEDs would have to have to predict a lifetime of 13.6

days. 2000 mA hours would last 13.6 days with a current consumption of 6.1 mA, so this is the total current drawn during this experiment. If the other duty cycles were guessed correctly, the figure for the LEDs must be 5.9 mA as shown in Figure 16.

Part	Active	Current in mA
Processor	1.67%	0.141
Radio (Sleep)	0.00%	0.002
LEDs (3 On)	100%	5.900
Flash Memory	0.21%	0.033
Sensor	0.42%	0.026
Total		6.102
	Hours	327.71
	Days	13.65
	Months	0.46

Figure 16: Revised LED current consumption

The current for this experiment when measured with an amp meter is 7.74 mA, giving an expected lifetime of 258.56 hours or 10.77 days. This would indicate that the experiment should have finished 43 hours before it actually did. The fact that it continued operating down to such a low voltage probably helped in achieving this.

3.2.6 Experiment 5: Continuous CPU Activity

For the continuous CPU experiment, the radio was turned off and no sensor board was fitted. When the mote was instructed to start this experiment a second timer was started on a millisecond interval. Each time the millisecond timer event was triggered, the processor counted to one hundred. This was intended to keep the CPU awake and prevent it entering a sleep state. The calculation of the expected power consumption and life expectancy is shown in Table 9.

Part	Active	Current in mA
Processor	100%	8.000
Radio (Sleep)	0.00%	0.002
Flash Memory	0.21%	0.033
Sensor	0.42%	0.026
Total		8.061
	Hours	248.09
	Days	10.34
	Months	0.34

Table 9: Continuous CPU activity estimated duration

Just having the processor 100% active should give a power consumption of 8.06 mA giving an estimated lifetime of 10.34 days or 248 hours. Experiment H stopped after 793.7 hours of operation and experiment I stopped after 794.8 hours.

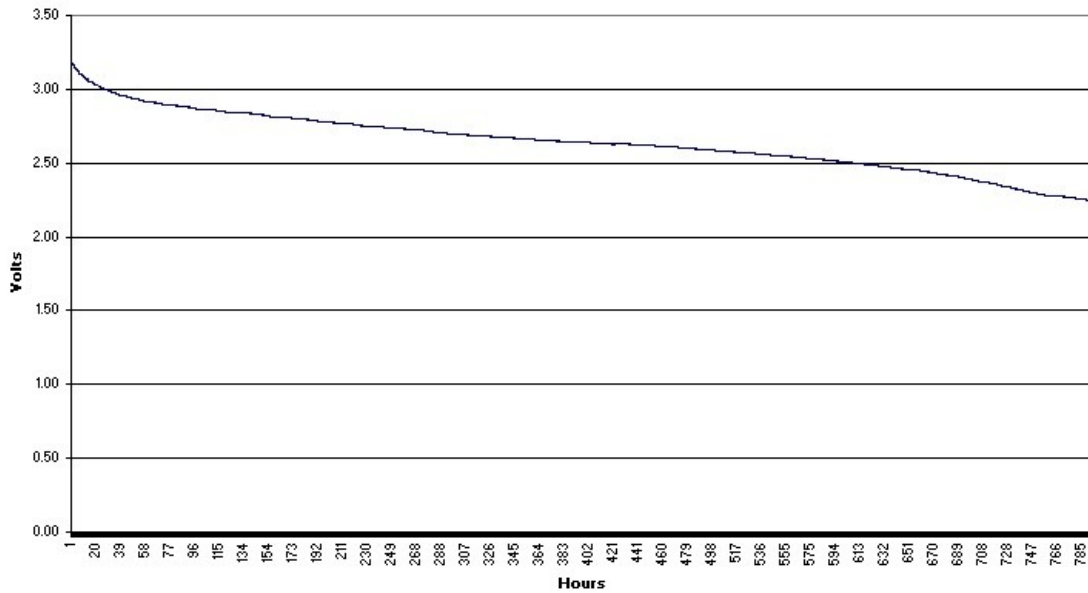


Figure 17: Exp H, Voltage Drop CPU Only

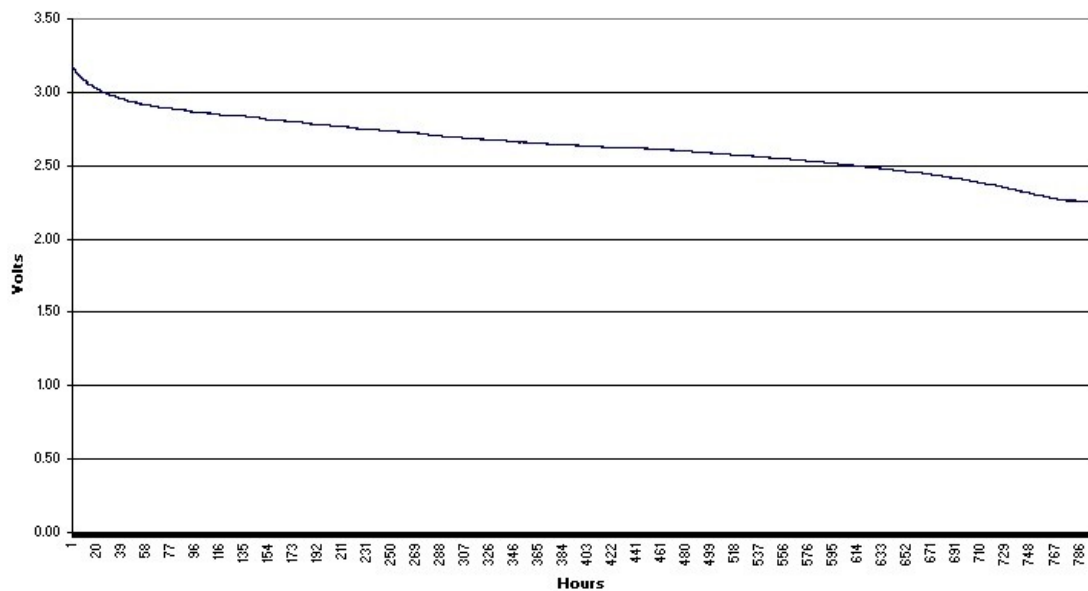


Figure 18: Exp I, Voltage Drop CPU Only

Both experiments continued long past the expected duration and operated down to 2.25 Volts. The current for this experiment when measured with an amp meter is 3.13 mA, giving an expected lifetime of 638.98 hours or 26.62 days. Assuming that the

processor does draw 8 mA when active, this shows that the steps taken to keep the CPU 100% accurate were not successful. The duty cycles were therefore overestimated.

3.2.7 Experiment 6: Continuous Voltage Sensing

Two different continuous sensing experiments were carried out, one to measure voltage and the other to measure light. The voltage can be measured on any Mica2 mote, but to measure light an additional sensor board must be fitted to the mote. Although the documentation does not specify different power consumption for each, both were tested. For the voltage sensing experiment the radio was turned off and no sensor board was attached. When the mote is instructed to start the experiment it takes a reading of the voltage. Reading any sensor in TinyOS is a split phase operation. After taking the reading control is passed to the operating system. When the reading has been taken an event is triggered to give the result. Motes running this experiment used this triggered event to order another reading, thus leading to cyclic readings. The calculation of the expected power consumption and life expectancy is shown in Table 10,

Part	Active	Current in mA
Processor	100%	8.000
Radio (Sleep)	0.00%	0.002
Flash Memory	0.21%	0.033
Sensor	100%	5.000
Total		13.035
	Hours	153.43
	Days	6.39
	Months	0.21

Table 10: Continuous Sensing estimated duration

As the CPU is used to start the reading and is immediately activated after the reading, the expected CPU activity is 100%. The gap between readings is very low, so the expected sensor activity is also 100% for this experiment. These figures together with the previous measurement figures give an estimated power consumption of 13.03 mA and an estimated lifetime of 6.4 days or 153 hours.

Only one experiment was carried out on the voltage sensor. Experiment J continued for 341.8 hours, 122.8% longer than expected. The voltage drop of the voltage sensing experiment is shown in Figure 19.

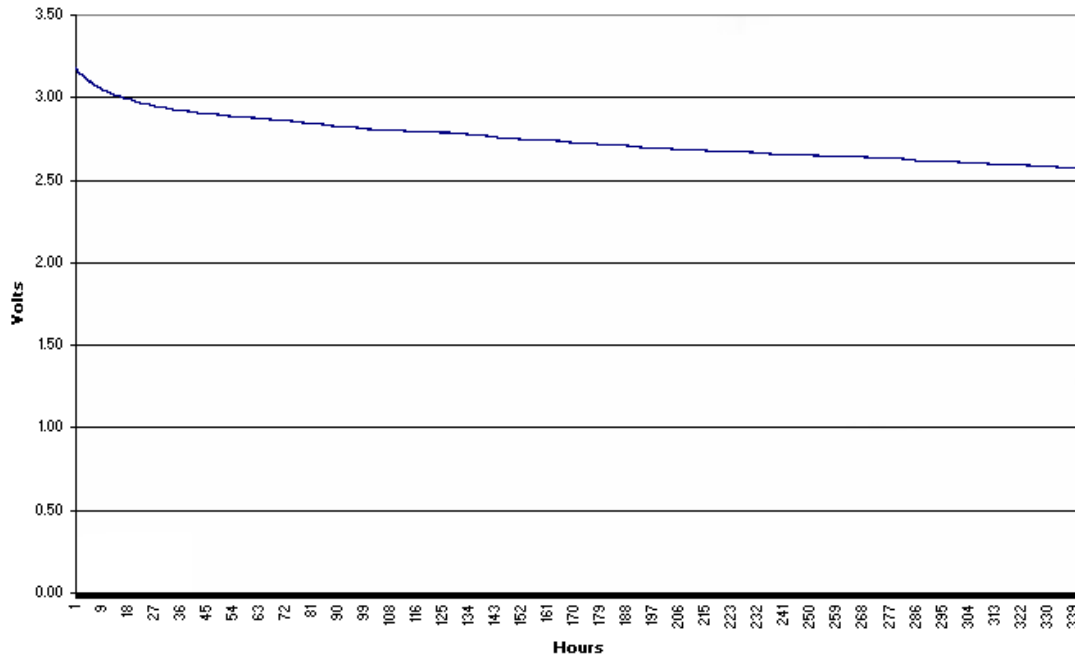


Figure 19: Exp J, Voltage Drop Sensing Voltage

Although the experiment continued far beyond the expected duration, the results show that it ceased operation at 2.57 Volts. The voltage at failure is higher than any other experiment. All other experiments, which were had time to terminate, continued to operate beyond 2.5V.

The current for this experiment when measured with an amp meter is 4.77 mA, giving an expected lifetime of 419.29 hours or 17.47 days. This shows that the duty cycles were overestimated.

3.2.8 Experiment 7: Continuous Light Sensing

This experiment was the same as the continuous voltage sensing experiment, but the light sensor was being constantly sampled rather than the voltage sensor. In addition to the constant light sensing, a voltage sample was taken at the end of each minute as normal. In respect of power consumption, nothing has changed, so the expected power consumption is still as shown in Table 10 at 13.03 mA giving an estimated lifetime of 6.4 days or 153 hours.

As sensing light requires the addition of a sensor board and more components, one would expect a higher current consumption and a shorter duration. In fact the light sensing experiments continued longer than the voltage sensing experiment and measuring with an amp meter shows that it draws less current.

Experiments K and L had to be stopped after 589.5 hours and 407.5 hours respectively. The voltage in experiment K reached 2.5 Volts, but the voltage in experiment L when it was terminated was 2.64 Volts. The voltage drop for these experiments are shown in Figures 20 and 21.

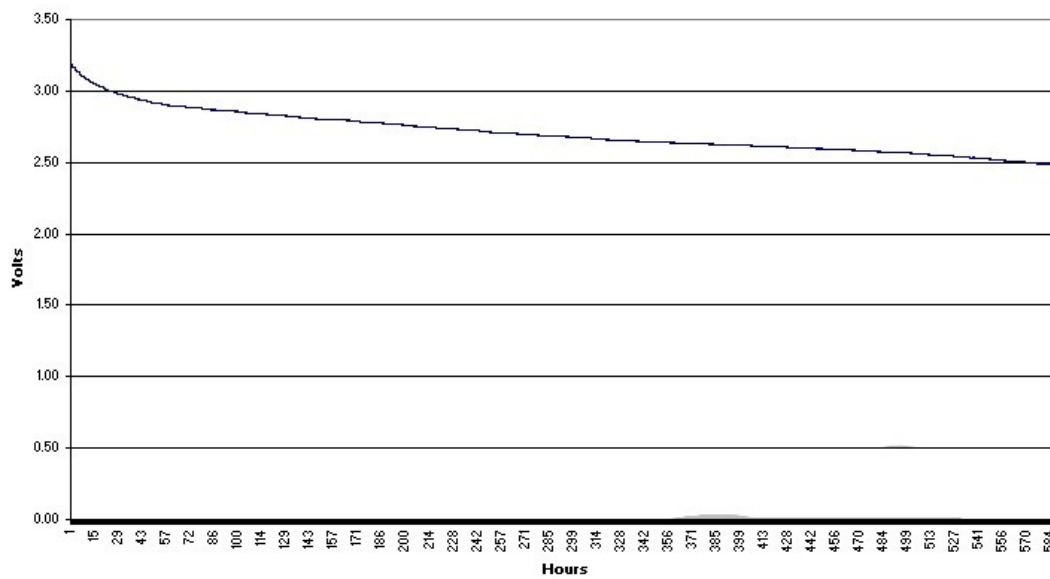


Figure 20: Exp K, Voltage Drop Sensing Light

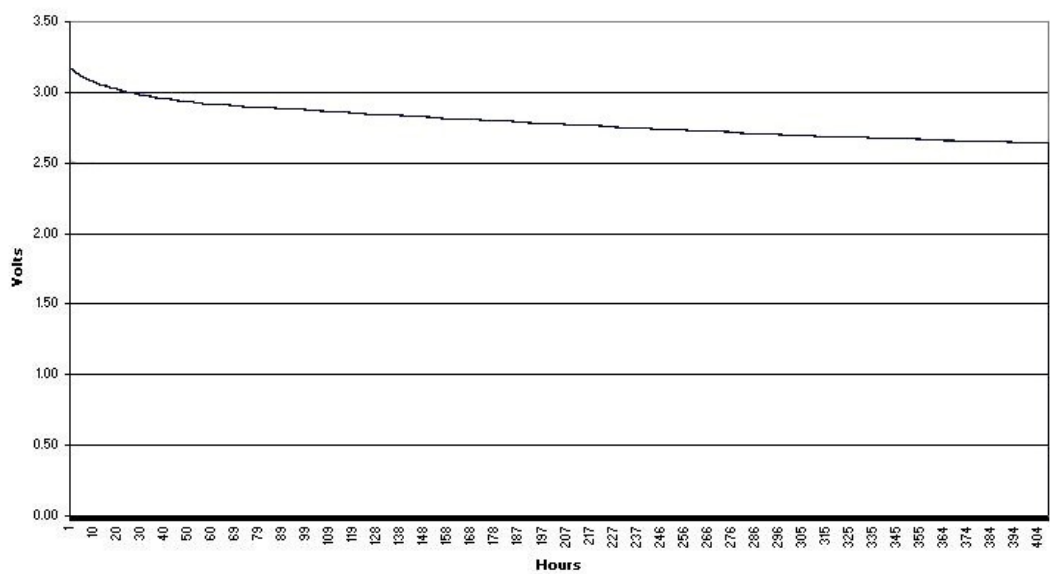


Figure 21: Exp L, Voltage Drop Sensing Light

The current for this experiment when measured with an amp meter is 3.43 mA, giving an expected lifetime of 419.29 hours or 17.47 days. This shows that the duty cycles were overestimated.

3.2.9 Overall Result

It is easier to evaluate the voltage drop when all of the experiments are plotted on one graph as in Figure 22.

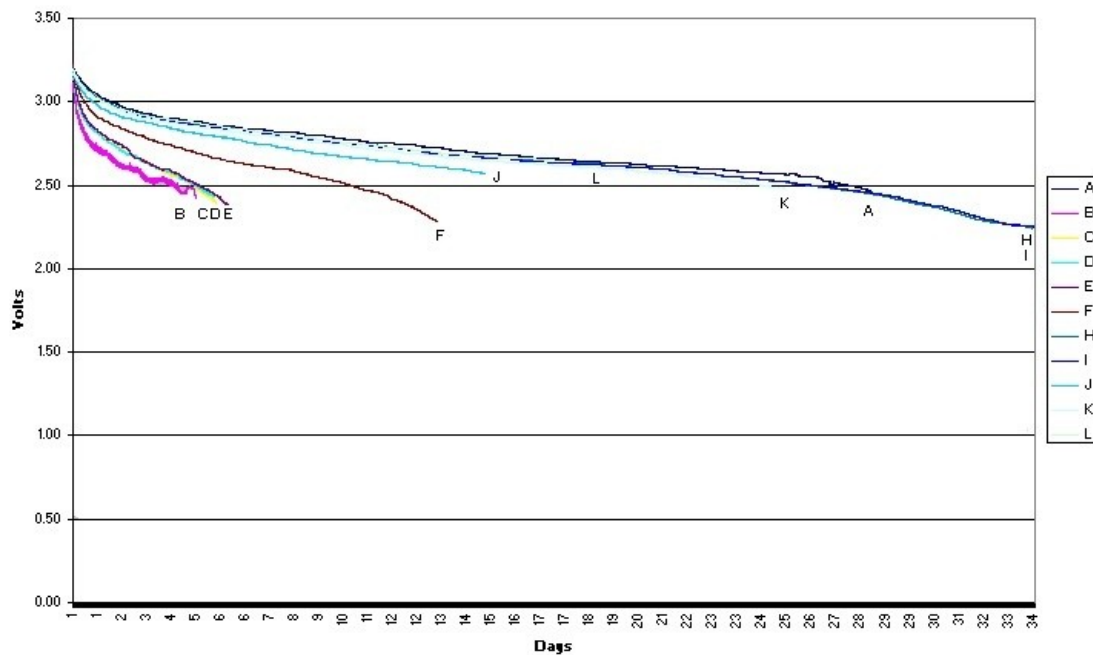


Figure 22: Voltage Drop for all Experiments

Here we can see that voltage sensing (J), is the only full term experiment terminating above 2.5 Volts. Experiment L had to be stopped due to time constraints. All of the experiments with the radio on were the shortest in duration. Experiments B, C, D and E all lasted less than a week. Next shortest in duration was the LEDs experiment (F), lasting less than a fortnight. Experiments A, K and L had to be stopped, so it is not known how long they would have continued to operate. The CPU only experiments (H and I) were the longest running, terminating after 34 days.

3.2.10 Current Consumption Estimation

Clearly in some of the experiments the estimated power consumption was inaccurate. This is due to overestimation and underestimation of some of the duty cycles. After the experiments had concluded, the current drawn in each experiment was checked

with an amp meter and compared with the original estimation. The results are shown in Figure 23.

Experiment	Estimated mA	Measured mA
No Operation	0.202	2.92
Continuous Signal Transmission	24.859	16.94
Continuous Signal Reception	16.059	15.75
LEDs On	8.202 / 6.102	7.74
Continuous CPU Activity	8.061	3.13
Voltage Sensing	13.035	4.77
Light Sensing	13.035	3.43

Figure 23: Estimated v. Measured Power

This shows that estimating duty cycles to predict current consumption is error prone without a method for measuring cycles accurately.

3.3 Predicting Remaining Battery Life

Knowing the manner in which the battery drains helps devise a method for predicting the point of failure and the number of hours remaining. Two possible methods for doing this are considered.

1. Once the voltage level has been recorded at fixed intervals over a given time, the average loss per interval can be calculated and from this an estimate of when the voltage will fall below a certain level can be derived. This is referred to as the linear prediction, as it is equivalent to finding the slope of the line so far and projecting it until it crosses a given point.
2. Given that the current drain of the various devices while being used or sleeping is known, the length of time each device spends in each state can be recorded and then the same calculation as before can be used to estimate the remaining battery life. This is referred to as the power method.

First we must determine the point at which the battery should be considered to be past its useful life. Clearly from the previous tests, the Mica2 motes carry on long after the operating voltage has been passed. As all but one of the experiments, which ran to full term, operated below 2.5 Volts, this was selected as the level to predict. In most graphs, the mote crosses the 2.5 Volt line shortly before it stops operating.

These methods are applied to the collected data to test the accuracy of predicting when the battery level will drop to 2.5 Volts.

Initially the voltage of the battery decreases rapidly in all experiments and then flattens out becoming more linear until the end. The linear prediction therefore will be very inaccurate at the beginning, but as the line levels out the prediction becomes more accurate.

The power prediction does not depend on previous measurements. The remaining lifetime is calculated based on the battery level and the current usage cycles of the various parts of the mote. The calculation is made in the same manner as the estimates for the voltage drop experiments. The remaining mA hours left in the battery is calculated from 2000 mA hours at 3.2 Volts down to 0 mA hours at 2.5 Volts. In the absence of a suitable equation to match the observed discharges, a linear discharge was assumed.

To check the accuracy of these methods, the calculation was made at each point recorded during the experiments to predict when the battery level would drop down to 2.5 Volts. The deviance from the actual time when this occurred in minutes is then calculated and plotted for each minute of the experiment. The voltage sensing experiment did not run until the battery discharged as far as 2.5 Volts, so no comparison with the actual time that this occurred is possible.

3.3.1 Experiment 1: No Operation

A plot of the error in predicting when 2.5 Volts is reached in this experiment is shown in Figure 24.

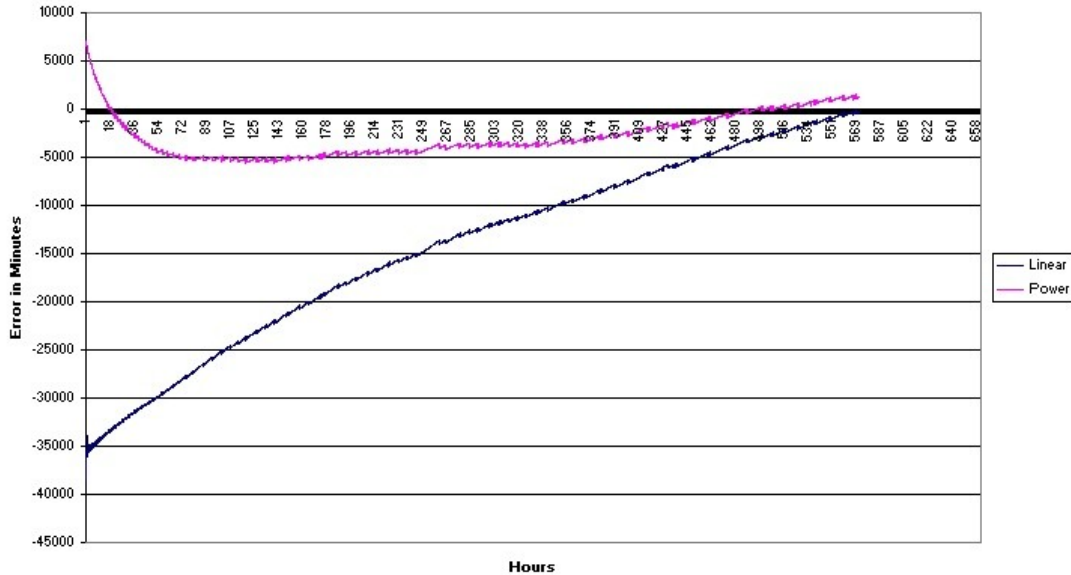


Figure 24: Exp A, Error Predicting 2.5 V Time

In the linear prediction, we see that soon after the experiment started the predicted time when 2.5 Volts would be reached was approximately 35,000 minutes less than when it actually occurred. This is due to the sharp decline of battery voltage at the beginning of the experiment. As time passes the prediction gets more accurate until 2.5 Volts is reached.

For the power prediction, the measured power consumption was used in all experiments. For this one that figure is 2.92 mA. The prediction starts off being overestimated by about 5,000 minutes of the actual time, but changes to an underestimation and starts to converge on the correct answer. Towards the end the time is overestimated again. The shape of this curve is due to the assumption that the discharge is linear, when in fact it is not.

3.3.2 Experiment 2: Continuous Signal Transmission Prediction

Plots of the error in predicting when 2.5 Volts is reached in Experiments B and C for continuous signal transmission are shown in Figure 25 and Figure 26 respectively.

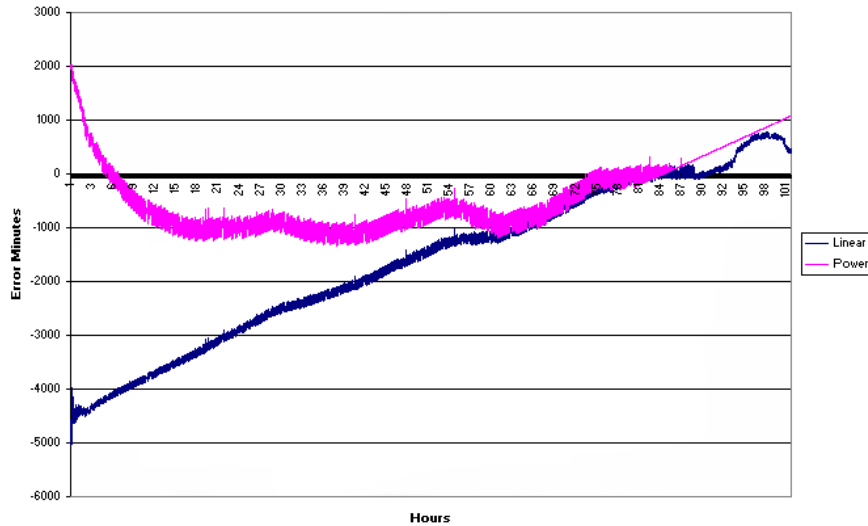


Figure 25: Exp B, Error Predicting 2.5V Time

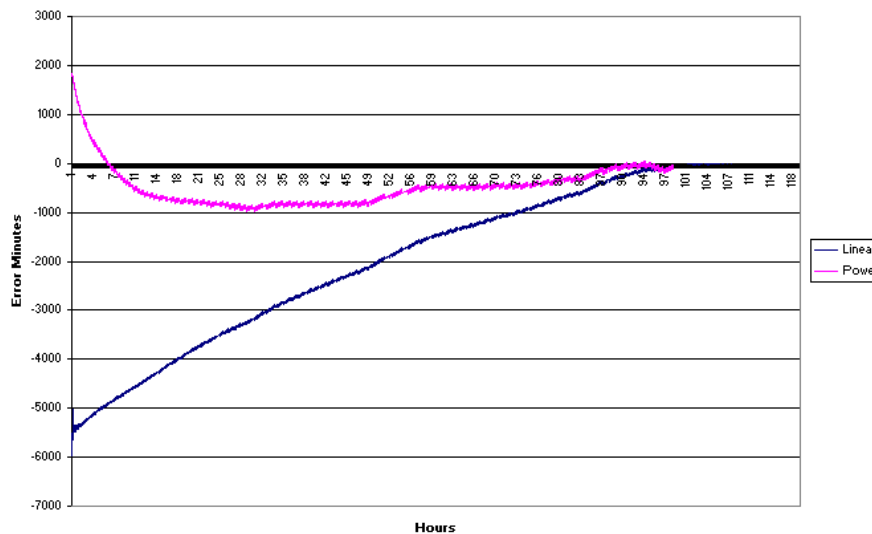


Figure 26: Exp C, Error Predicting 2.5V Time

In the linear prediction for each of these experiments, we see that soon after the experiment started the predicted time underestimated by more than 5,000 minutes. As previously, the prediction gets more accurate as time passes until 2.5 Volts is reached. For the power prediction, the measured power consumption of 16.94 mA was used in the calculation. This prediction starts off overestimating by 2,000 minutes of the actual time, but increases in accuracy and then underestimates and starts to converge on the correct answer.

3.3.3 Experiment 3: Continuous Signal Reception Prediction

Plots of the error in predicting when 2.5 Volts is reached in Experiments D and E for continuous signal reception is shown in Figure 27 and Figure 28 respectively.

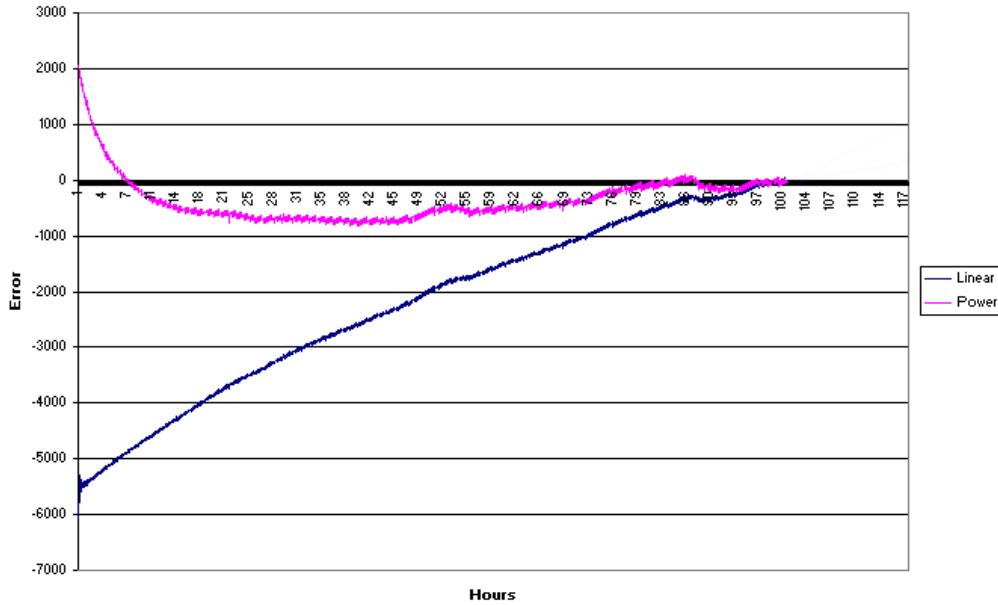


Figure 27: Exp D, Error Predicting 2.5V Time

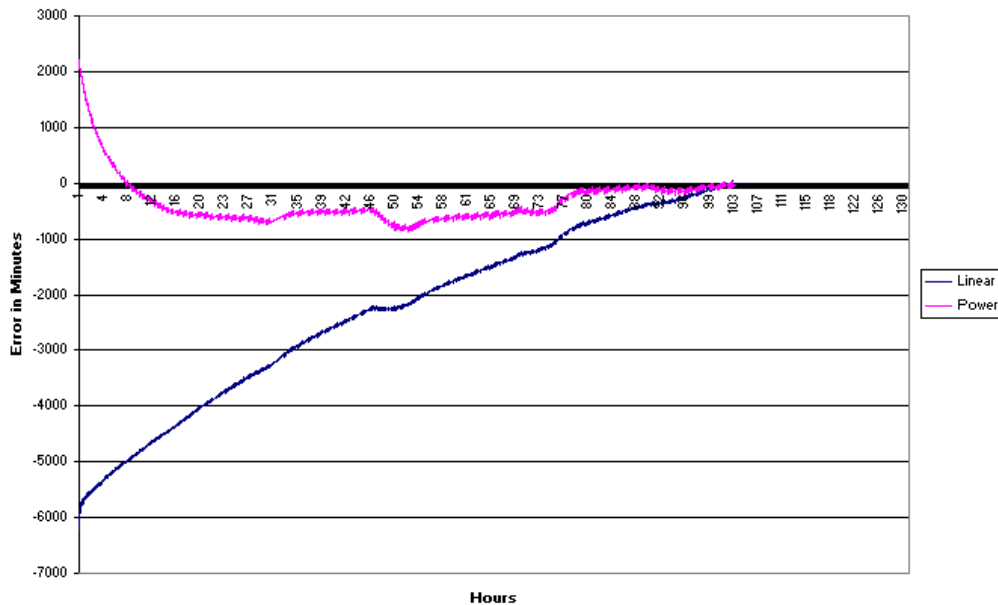


Figure 28: Exp E, Error Predicting 2.5V Time

The linear projection here is as one would expect. The power calculation in this case is overestimated by 2,000 minutes initially, but following this remains within 1,000 minutes of the correct time while approaching 2.5 Volts. Our calculation of the battery lifetime was most accurate for these experiments.

3.3.4 Experiment 4: LEDs Always On Prediction

A plot of the error predicting when 2.5 Volts is reached, using the measured current of 7.74 mA, for continuous LED operation is shown in Figure 29.

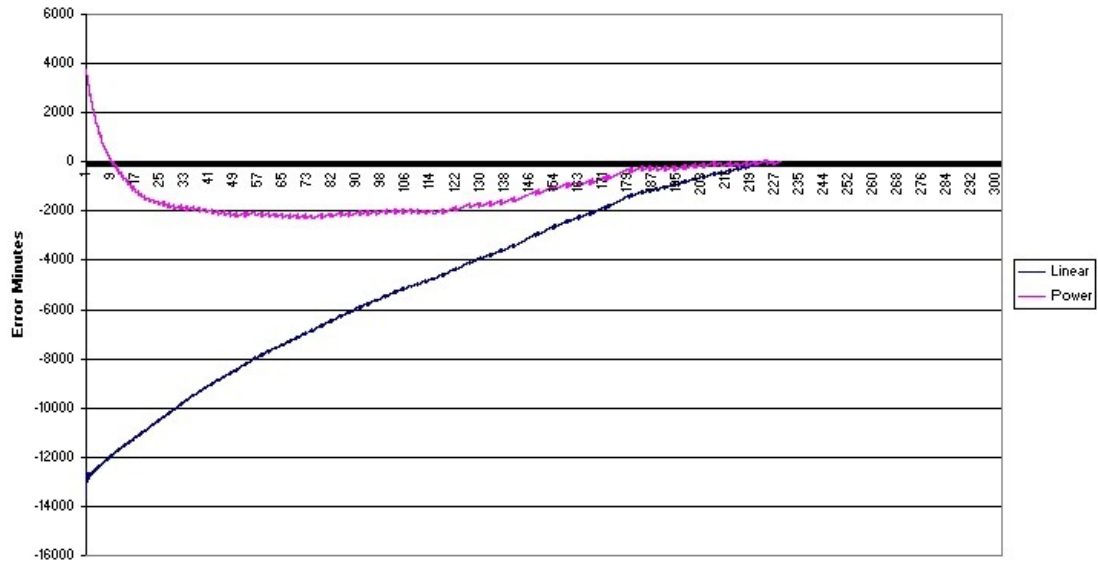


Figure 29: Exp F, Error Predicting 2.5 V Time

Again the linear prediction is as expected. The power calculation again starts with an inaccurate prediction of almost 4,000 hours overestimated and decreases this down to underestimating by over 2,000 before getting to an accurate estimation.

3.3.5 Experiment 5: Continuous CPU Activity Prediction

Experiment I and J ran for the same length of time and had almost identical results. A plot of the prediction error is shown in Figure 30.

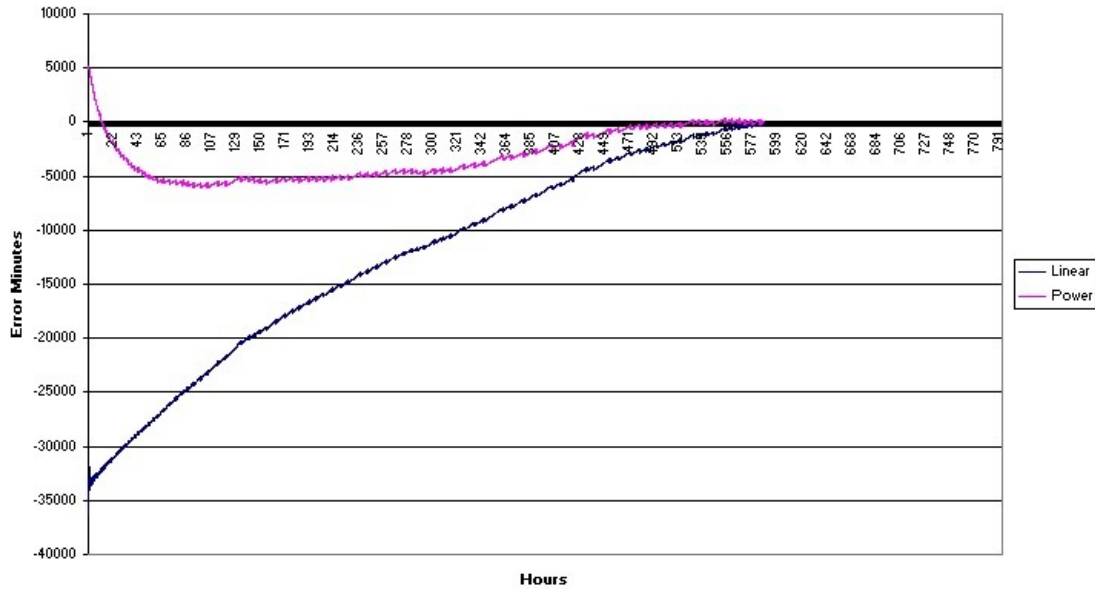


Figure 30: Exp H, Error Predicting 2.5 V Time

The linear prediction error is as expected. The margin of error was quite large on the power calculation, overestimating by 5,000 minutes at the start and then underestimating by a similar amount before approaching the correct answer.

3.3.6 Experiment 7: Continuous Light Sensing

The light sensing experiments had to be turned off due to time constraints. In only one experiment (K) the final voltage was lower than 2.5 Volts. The results for plotting the error in predictions for experiment K is shown in Figure 46.

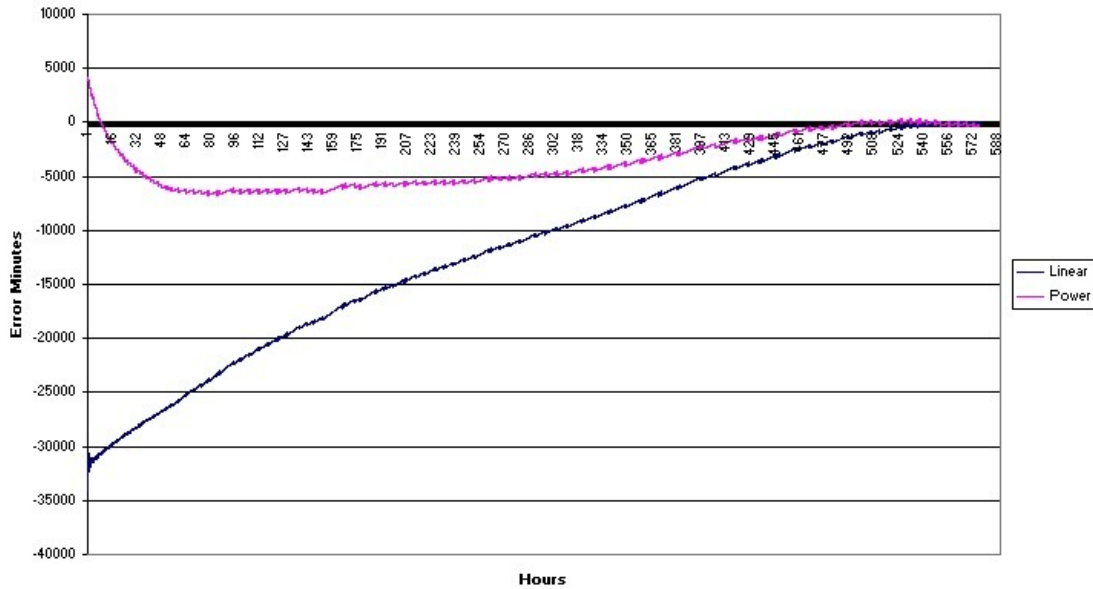


Figure 31: Exp K, Error Predicting 2.5 V Time

The linear prediction error is as expected. The current of this experiment (3.43 mA), when measured, was very similar to that of experiment H (3.13 mA). The power prediction therefore shows a similar pattern. At the start there is an overestimation of almost 5,000 minutes which changes to an underestimation of a similar amount before approaching the correct answer.

4 Implementation Details

The aim was to make a mote aware of its own lifetime and predict how long it will continue to function in the current situation. This enables a mote to make decisions based on the remaining lifetime. This section describes how this was implemented on the Mica2 motes.

4.1 Self Preservation Module

The functionality for this project was implemented in a TinyOS component called SelfP (Self Preservation). It was implemented as a component so that it can be easily wired into any TinyOS application.

As the effectiveness of the power calculation approach gave better results, this method was used for implementation. The power calculation requires the mote to know how long each part of the device spends in each state. The implementation of this was done using a timer, on a very short interval, to increment a counter, which was then be used to measure what proportion of time in a given interval was spent doing various activities. For this functionality to work, the module needs to mark the points at which changes in state take place. For this purpose commands in the component are used to mark when various devices are being woken up and put back to sleep. The start time spent in each mode is recorded and can be used at given intervals to calculate the percentage of each duty cycle. The result of this calculation is used to estimate current power consumption and remaining battery life. During debugging, it was advantageous to have the remaining hours and other variables calculated available to calling modules through the component interface. This was left in place as it could useful to applications.

It is envisaged that most applications would want to stop responding to signals as the battery approaches the end of its useful life. The signals that the mote should stop responding to are application specific. To provide the greatest flexibility, a rule-based approach was used for implementing decision-making logic.

4.1.1 SelfP Interface

The interface of the SelfP module does not contain any events. All functionality is available through the following commands.

command setPwr (newLvl)

This command is used to set the current power level to the value read from the ADC. As the power level can change at different rates depending on the application, this allows the application to decide how often the sensor is read. SelfP does not read the sensor directly, but uses this value to calculate remaining lifetime.

command stop ()

This command was added to aid debugging. When this command is called, SelfP stops measuring and all variables are frozen. Signalling is used to obtain variable values from the mote, so it is necessary to turn the measuring off prior to obtaining the information, as signalling affects the value of variables.

command setRadio (val)

This command must be used to notify the component whenever the Radio changes state. The values passed with this command have meanings described in Table 11.

Value	Meaning
0	Radio has been switched off
1	Radio has been switched on (listening)
2	Radio is receiving
3	Radio is transmitting

Table 11: Values for setRadio command

command setProc (val)

This command must be used to notify the component whenever the Processor has or is about to change state. The values passed with this command are described in Table 12.

Value	Meaning
0	Processor is being switched off
1	Processor has been switched on

Table 12: Values for setProc command

command setSensor (val)

This command must be used to notify the component whenever the Sensor changes state. The values passed with this command are described in Table 13.

Value	Meaning
0	Processor is being switched off
1	Processor has been switched on

Table 13: Values for setSensor command

command setLog (val)

This command must be used to notify the SelfP component whenever the flash memory has changed state. The values passed with this command are described in Table 14.

Value	Meaning
0	Flash Memory is idle
1	Reading to flash memory
2	Writing to flash memory

Table 14: Values for setLog command

command hrsLeft ()

This command returns the estimated number of hours of battery life remaining. Provided the setRadio, setProc, setSensor and setLog commands have been called at the appropriate points the component knows how long each part of the system has spent in various modes. This allows it to calculate the current power consumption at regular intervals. This in conjunction with the ADC reading stored with the setPwr command allow it to return an estimate of the number of hours of battery life.

command getVar (var)

This command allows an application to retrieve information about various parameters used in the calculation, along with any set user variables. The meaning of the value returned from this command varies according to the value passed with the command as given in Table 15.

Value	Return value contains
0	Set power level
1	Radio on (listening) Ratio * 100
2	Radio Receive Ratio * 100
3	Radio Transmit Ratio * 100
4	Current Radio Status (as in Table 11)
5	Sensor On Ratio * 100
6	Current Sensor Status (as in Table 13)
7	Log Read Ratio * 100
8	Log Write Ratio * 100
9	Current Log Status (as in Table 14)
10	Processor On Ratio * 100
11	Current Processor Status (as in Table 12)
12	Measuring Interval in Minutes, how often the current power consumption is estimated. (Default 2, Max 69905)
13-19	User variables
100	Internal SelfP time
101	Sensor Start Time
102	Radio Current State Start Time
103	Log Current State Start Time
104	Processor Current Start Time
105	Radio On Total Time this period
106	Radio Receive Total Time this period
107	Radio Transmit Total Time this period
108	Log Read Total Time this period
109	Log Write Total Time this period
110	Reserved
111	Processor On Total Time this period
112	Sensor On Total Time this period
113	Calculated current micro Amps
114	Is SelfP timer running
115	μ A calculated in last period
116	μ A calculated in last period + 1
117	μ A calculated in last period + 2

Table 15: Values for getVar command

The user variables are used in the implementation of rule processing. If an application wants to stop responding to a signal once the voltage drops below a certain level, that level is specified in a user variable. A rule can then be added to stop processing the signal when the voltage drops below that level.

Most of the variables above the user variables were primarily used for debugging purposes. Variables 115 to 117 however can be used to detect sudden changes in current consumption. The number of periods remembered is 3 by default, but can be changed in the header file. Increasing this number makes extra variables available.

The number of user variables available can also be adjusted in the header file up to a maximum of 88 without altering the code.

command setVar (index, val)

This command allows an application to set a user variable. The purpose of the parameters is given in Table 16. The set values can be used in the user-defined rules. All variables defined in Table 15 are read-only with the exception of 12, which is used to alter the measuring interval, and the user variables. Most of the other variables are reset at the end of each measuring interval.

Parameter	Purpose
index	The number of the variable to change
val	The new value

Table 16: setVar parameters

command addRule (msgType, indx1, cond, indx2)

This command is used to add a rule to the set of rules. It specifies the conditions under which a received signal will not be processed. The purpose of the parameters is given in Table 17.

Parameter	Purpose
msgType	The signal to ignore if the rule is true
indx1	The index of the 1 st variable to compare
cond	The comparison condition 1: Check $\text{indx1} == \text{indx2}$ 2: Check $\text{indx1} != \text{indx2}$ 3: Check $\text{indx1} < \text{indx2}$ 4: Check $\text{indx1} \leq \text{indx2}$ 5: Check $\text{indx1} > \text{indx2}$ 6: Check $\text{indx2} \geq \text{indx2}$
indx2	The index of the 2 nd variable to compare

Table 17: addRule parameters

Each signal in TinyOS has a number associated with it, usually specified in the application header file. If an application wants to ignore a signal in a certain situation it calls this command passing the number of the signal in the msgType and the condition in which it is to be ignored in the remaining parameters. For example to ignore signal 6 if the power measured on ADC channel 7 has a value greater than the value stored in user variable 14 and application would code as follows:

```
call SelfP.addRule (6, 0, 5, 14);
```

The command returns true if the rule was added, or false if there was no more room. The number of rules that can be specified can be specified in the header file.

command ok2do (msgType)

This command is used to check if a signal should currently be processed. The number of the signal is passed as a parameter and the signal returns true if the signal should be processed, otherwise it returns false. In the absence of any rules this command will always return true. Signals are processed unless any rule specifies that it should not be processed.

4.2 KillMote with SelfP

The SelfP component was wired into the KillMote application developed for running the experiments. This allows a comparison between the estimated duty cycles used for predicting the duration of the experiments with the duty cycles measured within the mote. The SelfP component was wired into the KillMote application following the normal TinyOS procedures. Calls were then made to the various commands throughout the application to mark when the radio and other devices were being used. Code was then added to the mote so that after a time long enough to make some measurements the experiment was stopped, SelfP was stopped, a LED was lit and the radio turned back on if it had been off. The user interface could then be used to extract information about duty cycles and estimated hours remaining.

Calling from the application level prevented capturing the duty cycles of the processor. The SelfP component therefore always calculated with 100% processor cycles. Reading the sensor and writing to the flash memory each minute counted up some milliseconds, but not enough to register in a percentage to 2 decimal places. Values less than 300 milliseconds are less than 0.005% of a minute and are regarded as zero for the calculation. For the transmit experiment, SelfP measured the radio in 100% transmit mode as expected, but for the receive experiment the radio was measured as 100% on, but not receiving. This occurs because signals are received and buffered before the event in the application is called. The signal reception has already taken place before the application is informed about it. In the sensing experiments, the sensor was marked as active for 75% of the time, rather than 100% as expected. Further research is needed to verify that the measured cycles for the sensor and flash memory are accurate. The power calculations made in the component are shown against the actual measured power in Figure 32.

Experiment	Estimated mA	Measured mA
1. No Operation	8.00	3.70
2. Continuous Signal Transmission	24.80	17.81
3. Continuous Signal Reception	16.00	16.64
4. LEDs On	8.00	8.54
5. Continuous CPU Activity	8.00	3.86
6. Voltage Sensing	11.75	4.85
7. Light Sensing	11.75	3.93

Figure 32: Power consumption calculated by SelfP

There is an overestimation of CPU activity in all the results. The calculation did not take any account of LEDs being on, so the fact that the estimate for experiment 4 is almost correct is due to this overestimation. Although LED duty cycles are not usually considered in calculations, they should clearly be taken into account if they are to remain on for any length of time. The current used by the three LEDs being lit is approximately 4.84 mA as can be seen in the difference in measured power consumption between experiments 1 and 4. This does not equate to 1.61 mA per LED. The difference in current consumption between the state when the mote has the green LED lit and not lit is only 1.48 mA. Further experimentation is needed to obtain more accurate power levels for each LED.

4.3 Problems Encountered

The biggest problem for a newcomer to TinyOS is the length of time it takes to install the system and become familiar with the workings of TinyOS. TinyOS tutorials are available online, which guide beginners through the installation process and getting a supplied application compiled and running on a mote. The supplied application is a very simple one, which blinks the three LEDs at different frequencies so they appear to count in binary. There is a significant amount of work involved in just getting the Blink application up and running.

A quick tour of the TinyOS help web pages suggested that it was better to install TinyOS on a Linux machine. After installing Linux in a partition on the machine, the Blink application would not compile, so I installed the Cygwin application and tried that. I installed the MoteWorks software from Crossbow, which includes Cygwin and TinyOS 1.x. This was the first installation, which ran successfully. This also came with a tailored version of Programmers Notebook, a text editor with some macros defined for working within the Windows environment and calling Cygwin for compilation and transfer to the motes. After a number of attempts TinyOS 2 was compiling successfully under Cygwin. Since then a live CD option has become available on the TinyOS website. This gives a virtualised Linux with a complete TinyOS install and may help newcomers to get up and running sooner.

It took some time to become familiar with TinyOS. Having compiled and run the blink application on a mote, I found that I couldn't get the motes to communicate. A colleague told me that TinyOS comes configured for 433 MHz whereas ours are 915 MHz, thus saving me a lot of discovery time. NesC also has a few peculiarities such

as not being able to place code before variable declarations in functions and 16 bit variables having to be transferred into larger variables before performing operations, which otherwise truncate the result. Unfortunately these peculiarities were only found through debugging, which can be very slow in this environment.

Wiring the SelfP module into the KillMote application demanded a lot of changes to the application. One of the difficulties with this approach is that the application needs to notify the SelfP component each time the radio is started, the radio is finished or any other relevant action occurs. There was a difficulty in capturing the receive cycles from the application, as it is only notified after the signal has been received. In order to accurately capture the cycles, it is necessary to implement the calls deeper in the operating system. Another drawback of this approach resulted from the need to have accurate timing. A timer with a very short interval was used to increment a counter, so that the length of time spent in various activities could be measured accurately. In a practical application this could significantly affect current drain. The measured current drawn in the KillMote application after the SelfP module was added is shown in Figure 33.

Experiment	KillMote	with SelfP	Extra mA
No Operation	2.92	3.70	0.78
Continuous Signal Transmission	16.94	17.81	0.87
Continuous Signal Reception	15.75	16.64	0.89
LEDs On	7.74	8.54	0.80
Continuous CPU Activity	3.13	3.86	0.73
Voltage Sensing	4.77	4.85	0.08
Light Sensing	3.43	3.93	0.50

Figure 33: Extra power consumption of SelfP

5 Conclusions and Future Work

Working with the Mica2 motes and TinyOS has introduced me to an exciting new area of technology. On a number of occasions, since being introduced to motes, I have found myself thinking of how useful they could be for various everyday tasks like monitoring the temperature in the baby room or being notified when someone parks in my car space. I have no doubt that in some form motes will become commonplace in the not too distant future.

There is scope for further investigation and exploration in a number of areas.

- Accurate Cycle Measurements
- Integration with TinyOS
- Cycle Timing Methods
- Prediction Improvement
- Application Utilisation

The amp meter readings of the transmit experiment suggest that the processor and CPU were not active for 100% of the time. Further investigation is needed to discover how much of the time is actually spent in various modes. A more accurate technique for measuring duty cycles is required. This may be best achieved by direct monitoring of the radio hardware outputs e.g. Through the use of a spectrum analyser to detect actual transmit and silent periods.

The SelfP component was wired directly into the application. As a first attempt this gave reasonable results, but showed a need for greater accuracy in capturing duty cycles. Besides wiring the component into the application in the normal way, quite a few lines of code had to be added for informing SelfP when devices were being activated. This is a typical cross cutting concern and leads to application code, which is difficult to read, non-modular and error prone. The SelfP component needs to be wired into TinyOS at a lower level and become part of the operating system, giving applications access to the predictive and rule based parts without having to worry about whether duty cycles are being correctly captured.

Using a millisecond timer, while capturing changes accurately, introduces an extra current drain. Ideally the ability to predict battery life should not shorten battery life. The power consumption of the code executed by the timer itself is not significant, but if it significantly changes the duty cycle of the processor it could have more impact.

Further investigation is needed to establish how a longer time interval would affect calculations and what impact the timers have on processor sleep time.

Assuming a linear depletion of remaining power lead to errors in the prediction, especially at the start of operation, as the voltage drop is faster initially. A lookup table could be used to find the remaining mA hours for a given voltage. Better still curve fitting may provide a formula for creating a more accurate forecast.

Once a SelfP component has been thoroughly developed and tested work can start on examining how it can be utilised in different protocols and applications to extend battery life.

The problem of battery life and preserving it is recognised as a major problem not just for motes, but also for other portable electronic devices such as mobile phones, PDAs and laptops. More generally, a predictive component like SelfP, that can influence device behaviour based on the power characteristics, can find application in a wide range of devices and scenarios.

References

- [1] Atmel. “*ATmega128L Datasheet*”
- [2] Bokser V, Oberg C, Sukhatme G, Requicha A. “*A small submarine robot for experiments in underwater sensor networks*”, In Symposium on Intelligent Autonomous Vehicles, July 2004.
- [3] Carbajo R. “*Hopping: A Bidirectional Stigmergy Power Efficient On-Demand Driven Ad Hoc Routing Protocol for Wireless Sensor Networks*”, Master’s dissertation, University of Dublin, 2006.
- [4] Chandrakasan AP, Sheng S, Brodersen RW. “*Low-power CMOS digital design*”, Solid-State Circuits, IEEE Journal of, vol. 27, pp. 473-484, 1992.
- [5] Chang JH, Tassiulas L. “*Energy conserving routing in wireless ad-hoc networks*”, In Proc. IEEE INFOCOM, Tel Aviv, Israel, Mar. 2000.
- [6] Chen B, Jamieson K, Balakrishnan H, Morris R. “*Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks*”, In 7th Annual Int. Conf. Mobile Computing and Networking 2001, Rome, Italy, July 2001.
- [7] Chipcon / Texas Instruments. “*CC1000 Data Sheet*”.
- [8] Chlamtac I, Petrioli C, Redi J. “*Energy-conserving access protocols for identification networks*”, IEEE/ACM Transactions on Networking, 7(1):51-9, Feb. 1999.
- [9] Chockalingam A, Zorzi M. “*Energy efficiency of media access protocols for mobile data networks*”, IEEE Transactions on Communications, 46(11):1418-21, Nov. 1998.
- [10] Coleri S, Cheung SY, Varaiya P. “*Sensor networks for monitoring traffic*”, Forty-Second Annual Allerton Conference on Communication, Control, and Computing, Univ. of Illinois, Sept. 2004.
- [11] Cormen TH, Leiserson CE, Rivest RL, Stein C. “*Introduction to Algorithms, Second Edition*”, MIT Press and McGraw-Hill. ISBN 0-262-03293-7. Section 24.3: Dijkstra's algorithm, pp.595-601. 2001.
- [12] Crossbow Technology Inc. “*Mica2 Datasheet*”.
- [13] Crossbow Technology Inc. “*Micaz Datasheet*”.
- [14] Crossbow Technology Inc. “*MPR/MIB User Manual*”.

- [15] Dijkstra EW. “*A note on two problems in connexion with graphs*”, In *Numerische Mathematik*. 1, S. 269-271, 1959.
- [16] Doherty L, Warneke B, Boser B, Pister K. “*Energy and performance considerations for smart dust*”, *International Journal of Parallel Distributed Systems and Networks*, vol. 4, no. 3, pp. 121–133, 2001.
- [17] Dutta P, Grimmer M, Arora A, Bibyk S, Culler D. “*Design of a wireless sensor network platform for detecting rare, random, and ephemeral events*”, in *Proceedings of the Fourth International Conference on Information Processing in Sensor Networks (IPSN 2005)*.
- [18] Dutta PK, Culler DE. “*System Software Techniques for Low-Power Operation in Wireless Sensor Networks*”, *International Conference on Computer Aided Design, Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, San Jose, CA pp 925 - 932, 2005.
- [19] Ergen SC, Varaiya P. “*Tdma scheduling algorithms for sensor networks*”, *Technical Report, Department of Electrical Engineering and Computer Sciences University of California, Berkeley*, July, 2005.
- [20] Ganesan D, Greenstein B, Perelyubskiy D, Estrin D, Heidemann J. “*Multi-resolution storage and search in sensor networks*”, *ACM Transactions on Storage* Aug. 2005.
- [21] Gomez J, Campbell AT, Naghshineh M, Bisdikian C. “*Conserving transmission power in wireless ad hoc networks*”, presented at *Network Protocols*, 2001. *Ninth International Conference on*, 2001.
- [22] Gu L, Stankovic J. “*Radio triggered wake-up capability for sensor networks*”, in *Real-Time Applications Symposium (RTAS’04)*, May 2004.
- [23] Gupta P, Kumar PR. “*Critical power for asymptotic connectivity in wireless networks*”, *Stochastic Analysis, Control, Optimization and Applications: A Volume in Honor of W.H. Fleming*, pages 547 - 566, 1998.
- [24] He T, Krishnamurthy S, Stankovic JA, Abdelzaher T, Luo L, Stoleru R, Yan T, Hui J, Krogh B, Gu L. “*Energy-Efficient Surveillance System Using Wireless Sensor Networks*”, In *The Second International Conference on Mobile Systems, Applications, and Services (MobiSys)*, June 2004.
- [25] Hohlt B, Doherty L, Brewer E. “*Flexible power scheduling for sensor networks*”, In *Proceedings of the Third International Symposium on Information Processing in Sensor Networks*, Berkeley, CA, Apr. 2004.

- [26] Hui JW, Ren Z, Krogh B. “*Sentry-based power management in wireless sensor networks*”, The 2nd International Workshop on Information Processing in Sensor Networks (SPSN'03), 2003.
- [27] Jeong J, Culler D, Oh J-H. “*Empirical analysis of transmission power control algorithms for wireless sensor networks*”, Technical Report, University of California, Berkeley, UCB/EECS-2005-16, 2005.
- [28] Jin N, Venkitachalam G, Jordan S. “*Characteristics of Resource Allocation using Pricing*”, In Computer Communications, 2003. CCW 2003. Proceedings. 2003 IEEE 18th Annual Workshop on, pp. 59- 65, 2003.
- [29] Jin N, Venkitachalam G, Jordan S. “*Dynamic Pricing of Network Resources*”, In Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE, pp. 3216- 3220 vol.6, 2003.
- [30] Juang P, Oki H, Wang Y, Martonosi M, Peh LS, Rubenstein D. “*Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebrant*”, in Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X), 2002.
- [31] Kalpakis K, Dasgupta K, Namjoshi P. “*Maximum Lifetime Data Gathering and Aggregation in Wireless Sensor Networks*”, In Proceedings of the 2002 IEEE International Conference on Networking (ICN'02), Atlanta, Georgia, pp. 685-696. August 26-29, 2002.
- [32] Kulkarni SS, Arumugam M. “*Tdma service for sensor networks*”, In Proceedings of the Third International Workshop on Assurance in Distributed Systems and Networks, Mar. 2004.
- [33] Kulkarni SS, Arumugam U. “*Collision-free communication in sensor networks*”, In Proceedings of the Sixth Symposium on Self-stabilizing Systems (SSS), Springer ,LNCS:2704:17 - 31, June 2003.
- [34] Le L, Aikat J, Jeffay K, Smith DF. “*The Effects of Active Queue Management on Web Performance*”, In Computer Communication Review VOL 33; PART 4, pages 265-276, 2003.
- [35] Levis P, Madden S, Gay D, Polastre J, Szewczyk R, Woo A, Brewer E, Culler D. “*The emergence of networking abstractions and techniques in tinyos*”, In Proceedings of the First USENIX/ACM Symposium on Network Systems Design and Implementation (NSDI), 2004.

- [36] Li Q, Aslam J, Rus D. “*Online power-aware routing in wireless ad-hoc networks*”, In *Mobile Computing and Networking*, pp. 97 - 107, 2001.
- [37] Lin S, Zhang J, Zhou G, Gu L, He T, Stankovic JA. “*ATPC: Adaptive Transmission Power Control for Wireless Sensor Networks*”, In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2006.
- [38] Madden S. “*The design and evaluation of a query processing architecture for sensor networks*”, Ph.D. dissertation, U.C. Berkeley, 2003.
- [39] Maleki M, Dantu K, Pedram M. “*Power-aware source routing protocol for mobile ad hoc networks*”, presented at *Proceedings of the 2002 international symposium on Low power electronics and design*, Monterey, CA, USA, 2002.
- [40] Michail A, Ephremides A. “*Energy-Efficient Routing for Connection-Oriented Traffic in Wireless Ad-hoc Networks*”, *Mobile Networks and Applications* Volume 8, Issue 5 pp. 517 - 533, 2003.
- [41] Min R. “*Energy and Quality Scalable Wireless Communication*”, In *Department of Electrical Engineering and Computer Science*. Cambridge, MA: Massachusetts Institute of Technology, 2003.
- [42] Peukert W. “*Über die Abhängigkeit der Kapazität von der Entladestromstärke bei Bleiakumulatoren*”, *Elektrotechnische Zeitschrift* 20, 1897.
- [43] Polastre J. “*Design and implementation of wireless sensor networks for habitat monitoring*”, Master’s thesis, University of California at Berkeley, 2003.
- [44] Polastre J, Hill J, Culler D. “*Versatile low power media access for wireless sensor networks*”, In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys’04)*, Nov. 2004.
- [45] Rodoplu V, Meng TH. “*Minimum energy mobile wireless networks*”, In *Proceedings of the 1998 IEEE International Conference on Communications, ICC '98*, volume 3, pages 1633-1639, Atlanta, GA, June 1998.
- [46] Shih E, Bahl P, Sinclair MJ. “*Wake on wireless: An event driven energy saving strategy for battery operated devices*”, In *Proceedings of the Eighth Annual ACM Conference on Mobile Computing and Networking*, Atlanta, Georgia, USA. 2002.

- [47] Shih E, Cho S, Lee FS, Calhoun BH, Chandrakasan A. “*Design considerations for energy-efficient radios in wireless microsensor networks*”, The Journal of VLSI Signal Processing-Systems for Signal, Image, and Video Technology, vol. 37, no. 1, pp. 77–94, 2004.
- [48] Singh S, Raghavendra CS. “*PAMAS-Power Aware Multi-Access protocol with Signaling for Ad Hoc Networks*”, ACM Commun. Rev., July 1998.
- [49] Singh S, Woo M, Raghavendra CS. “*Power-aware routing in mobile ad-hoc networks*”, In Proc. of Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking, pages 181-190, Dallas, TX, Oct. 1998.
- [50] Stojmenovic I, Lin X. “*Power-aware localized routing in wireless networks*”, Parallel and Distributed Systems, IEEE Transactions on, vol. 12, pp. 1122-1133, 2001.
- [51] Werner-Allen G, Johnson J, Ruiz M, Lees J, Welsh M. “*Monitoring volcanic eruptions with a wireless sensor network*”, In Proceedings of the Second European Workshop on Wireless Sensor Networks (EWSN’05), Jan. 2005.
- [52] Woesner H, Ebert JP, Schlager M, Wolisz A. “*Power Saving Mechanisms in Emerging Standards for Wireless LANs: The MAC Perspective*”, Personal Communications, IEEE [see also IEEE Wireless Communications], vol. 5, pp. 40-48, 1998.
- [53] Woo A, Tong T, Culler D. “*Taming the underlying challenges of reliable multihop routing in sensor networks*”, In Proceedings of the first international conference on Embedded networked sensor systems, pages 14-27, 2003.
- [54] 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.
- [55] Xing G, Lu C, Zhang Y, Huang Q, Pless R. “*Minimum Power Configuration in Wireless Sensor Networks*”, ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc’05), May 2005.
- [56] Xu N, Rangwala S, Chintalapudi K, Ganesan D, Broad A, Govin-dan R, Estrin D. “*A wireless sensor network for structural monitoring*”, In Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys’04), Nov. 2004.

- [57] Xu Y, Heidemann J, Estrin D. “*Adaptive energy-conserving routing for multihop ad hoc networks*”, Research Report 527 USC/Information Sciences Institute, October 2000.
- [58] Xu Y, Heidemann J, Estrin D. “*Geography-informed energy conservation for Ad Hoc routing*”, In Proceedings of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking, Rome, Italy, July 2001.
- [59] Zorzi M, Rao RR. “*Error control and energy consumption in communications for nomadic computing*”, Computers, IEEE Transactions on, vol. 46, pp. 279-289, 1997.