A Wireless Sensor Network for Vibration Measurement

Shaun Kaplan

Thesis presented for the degree of
Master of Science
in the Department of Electrical Engineering
University of Cape Town

August 2011

Declaration

I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own.

Shaun Kaplan

Name Date

Abstract

Wireless sensor networks (sensornets) enable distributed sensing, opening up sensing possibilities not previously available. One application of sensornets is online, non-intrusive power transformer monitoring. Despite the high cost of transformer failures, power transformers often have little or no online monitoring. It has been shown that vibration monitoring is a suitable means of monitoring a transformer's health, as internal vibrations caused by magnetostriction and other sources propagate to the transformer tank. It has therefore been proposed that a sensornet be used to monitor power transformers by measuring vibration at various points on the transformer tank.

As such, a sensornet for distributed measurement and data logging is presented. This sensornet consists of three parts: sensor motes, a base mote and data logging software. The sensor motes and base mote are all TelosB motes programmed with TinyOS. The sensor motes are kept purposely simple, while control of the network is managed by the base mote. The base mote provides a global reference clock, requests sampling to begin, and recovers lost data packets. The flooding time synchronisation protocol is used for clock dissemination and timestamping.

The sensornet protocol was analysed through empirical tests and simulation. A sensornet of initially three, then later fifteen sensor motes was tested under controlled conditions to verify its operation. It is shown that applied frequencies and amplitudes are correctly represented, although amplitudes were less accurate at low input voltages. The analogue to digital converter samples each channel consecutively. As such, there is a delay between each sampled channel. This was shown to be approximately 22.765 μs, within range of the expected value. Flooding time synchronisation protocol timestamps enabled data from different motes to be synchronised in time, with an resolution of 30.5 μs. The sensor mote timestamps were typically within 1 to 2 clock cycles of each other. Phase comparisons between channels on different motes can be performed, but with limited accuracy due to the offsets between mote timestamps.

Additionally, the system was tested with accelerometers attached and operated as expected. Finally the sensornet was operated under conditions emulating the proposed application. Two networks were mounted on a three phase, 200 V isolation transformer, one network only measuring a single axis, with the other measuring three axes. These tests clearly showed the 100 Hz frequency component predominantly caused by magnetostriction in the core. Further work is required to interpret the measurements.

Acknowledgements

I would like to thank the following people:

- My supervisors: Fred Nicolls, Gerhard de Jager and Richardt Wilkinson for their guidance, assistance and support. It is greatly appreciated.
- Jevon Davies for his enthusiasm and MATLAB expertise.
- My parents and friends for their support.
- Gary Jacobson for proofreading.
- The CIR, and in particular, the denizens of room 5.19.
- All who have provided their support during this work.

Contents

List of Figures

List of Tables

List of Acronyms

A

ACK Acknowledgement.

ADC Analogue to digital converter.

ADCpp ADC sampled peak-to-peak values.

B

bps Bits per second.

C

CCA Clear channel assessment.

CCS Chaining clock synchronisation.

CODA Congestion detection and avoidance.

CSMA-CA Carrier sense multiple access with collision avoidance.

CSV Comma separated values.

CTP Collection tree protocol.

D

DIP Dissemination protocol.

E

ESRT Event-to-sink reliable transport.

F

FFT Fast Fourier transform.

FTSP Flooding time synchronisation protocol.

I

IFRC Interference aware fair rate control.

M

MAC Medium access control.

MEMS Microelectromechanical systems.

MSC Multiple sample and convert.

mVpp millivolts peak-to-peak.

N

NACK Negative acknowledgement.

NTP Network time protocol.

O

OLTC On-load-tap-changer.

P

PCB Printed circuit board.

PSFQ Pump slowly, fetch quickly.

R

RBS Reference broadcast synchronisation.

RCRT Rate-controlled reliable transport.

RFC Request for comments.

RMST Reliable multi-segment transport.

S

SPIN Sensor protocols for information via negotiation.

T

TCP Transmission control protocol.

TelosB Telos revision B.

TPSN Timing-sync protocol for sensor networks.

V

VA Volt-ampere.

W

WRAP Wireless reliable acquisition protocol.

Chapter 1

Introduction

1.1 Wireless sensor networks

Wireless sensor networks (sensornets) form a relatively new field that has grown rapidly through advances in microelectromechanical systems (MEMS), wireless technology and the availability of small, low power micro-controllers [13]. Sensornets are ad hoc networks of small, (supposedly) low cost, low power devices known as motes. Motes have sensing and sometimes actuating capabilities. These motes usually communicate using radio frequency (RF) communication, however optical motes have been proposed [89] as have acoustic motes for underwater sensornets [3].

Sensornets have been used or proposed for numerous applications. They are able to provide spatial data over large areas for long periods of time and are suited for many monitoring applications. These applications include monitoring of Leach's Storm-Petrels' nesting [55] and structural monitoring of a heritage building [10].

The challenges of sensornets are all tied to the motes having limited resources, most notably a limited power source. Motes are usually battery powered, with very few having access to mains power. Some motes use energy harvesting techniques to provide longer term operation. Solar energy and vibration are 2 potential sources of energy for motes [15]. Motes typically use low power micro-controllers and are therefore limited to the size and complexity of the firmware loaded. Common Internet protocols are not typically suited for use directly on motes due to the limited resources. Many sensornets use the IEEE 802.15.4 low-rate wireless personal area network standard that operates up to 250 000 bits per second (bps) and has a maximum data packet size of 127 bytes [37]. Routing, security, reliable data transfer, mote location and time synchronisation are some of the issues being tackled by the community.

A proposed application for a sensornet is to form part of a transformer condition monitoring system. It is envisioned that sensor motes will be placed on high power transformers and other electrical distribution equipment to non-intrusively monitor the state of the equipment. The motes will collect data and transfer this data to a gateway with more processing power and storage, and a connection to other computer networks. Processing may be performed in the sensornet, on the gateway and on remote computers. It is hoped that this system will be able to detect problems with the equipment to prevent failures from occurring. This application forms part of ongoing work in the Centre for Instrumentation Research (CIR) at the Cape Peninsula University of Technology (CPUT).

1.2 Transformer condition monitoring

The cost of unscheduled electrical network downtime is high, yet power transformers often lack instrumentation or have limited monitoring [72]. On-line monitoring of power transformers could provide sufficient information to enable effective preventative maintenance with minimal downtime.

Cardoso and Oliveira [9] present a summary of the causes of transformer failures with data from two transformer reliability surveys. They present the failure distribution and typical downtime, which is shown in Figure 1.1.

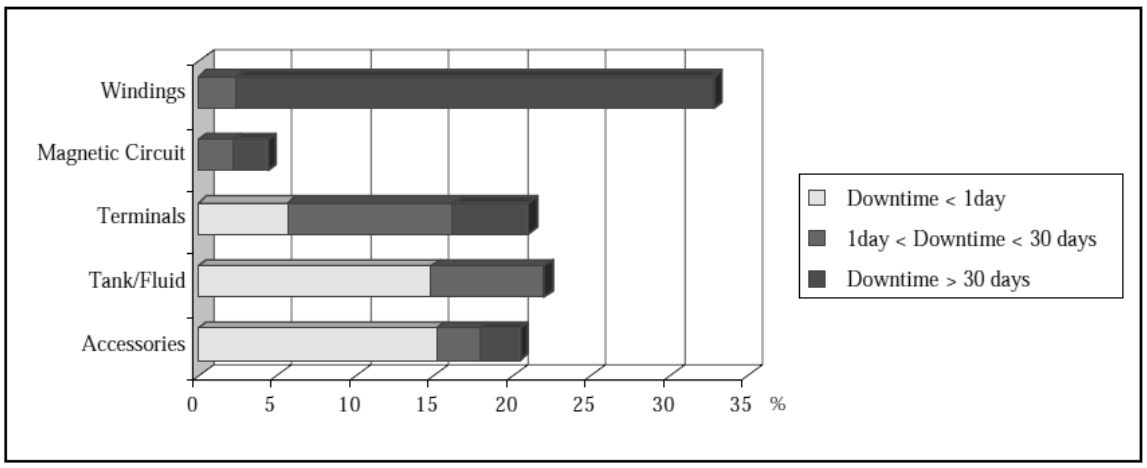


Figure 1.1: Distribution of transformer failure causes with typical associated downtime. Reprinted from [9].

It can be seen from the figure that the transformer windings are the most common and the most severe cause of transformer failure shown. Not shown in the figure, because it was only included in 1 of the 2 surveys, are failures caused by on-load-tap-changer (OLTC) faults [9]. Kang et al. [39] note that OLTCs contribute the majority

of transformer faults.

Abu-Elanien and Salama [1] present a survey on transformer condition monitoring. They list the 5 methods of transformer monitoring: thermal, vibration, dissolved gas, partial discharge and frequency response analysis. The authors conclude that further work is required for most methods to be used for condition monitoring.

We have chosen to begin our condition monitoring system using vibration measurements. Vibration analysis seems promising with a number of studies in the area [6, 25, 26, 67, 74]. Kang et al. have shown that vibration monitoring can be used to detect OLTC faults [39], while vibration monitoring can also be used to indicate winding and core problems [74].

The vibrations measured at the transformer tank surface are the result of a number of factors illustrated in Figure 1.2. The largest cause of vibration is magnetostriction in the core [7]. Magnetostrictive forces cause the laminations of which the core is comprised to vary in size with the changing magnetic field, i.e. at twice the supply frequency. The movement should be longitudinal, however, irregularities in the laminations, spaces between laminations, edge regions of the core, stresses from clamping and areas of differing permeability cause transverse movement [25, 22]. Magnetostriction is nonlinear with respect to flux density and so causes the vibration to include harmonics and not be a pure 100 Hz sine wave. The other factors mentioned worsen the harmonics.

Vibration of the transformer windings are caused by the interaction of leakage flux with the current in the windings. The winding vibrations are mostly dependant on current in the windings and therefore the loading of the transformer [25]. The vibrations from the core and windings are transferred through mechanical coupling and acoustically through the oil to the transformer tank. The construction of the tank further affects the vibration.

Core and winding vibrations can be determined in offline tests, though Shengchang et. al [74] propose a method of differentiating between core and winding vibration without placing the transformer in a no load condition.

1.3 Work presented

This thesis presents the design and implementation of a distributed data logger in the form of a sensornet. The sensornet was proposed to collect vibration measurements for analysis. This application required the selection of appropriate sensornet hardware

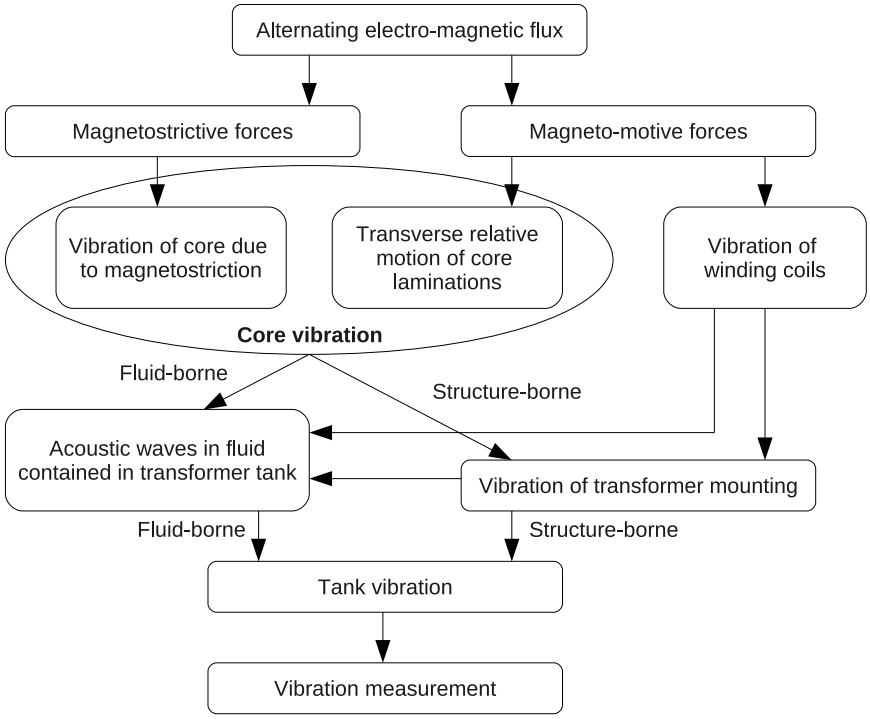


Figure 1.2: Vibration causes and propagation in a transformer. Adapted from [59].

and software. The widely used TelosB mote was chosen as the sensornet hardware platform with TinyOS as the embedded operating system. Vibration measurements were taken with LIS352AX 3-axis analogue accelerometers. These were chosen for the following electrical properties: low current consumption and suitable operating voltage range.

The sensornet consisted of three parts: sensor motes, a base mote and a data logger application. The sensor motes ran simple data sampling and forwarding software controlled by the base mote, which provided global time to the sensor motes, provided control packets to configure the sensor motes and tracked lost data packets to provide reliable communication. Global time was provided using the flooding time synchronisation protocol (FTSP). The data logger buffered all received data and wrote the data to comma separated values (CSV) files that could be imported into a spreadsheet or an analysis program like MATLAB. MATLAB was used to analyse and visualise the data.

The motes were required to be time synchronised to facilitate measurement of phase differences between spatially separated sensors.

Results show that the system operated as expected, though data packet losses were higher than results from simulations, with accurate reproduction of signals in the frequency domain, but with some amplitude inaccuracies at very small input voltages

in the time domain. Inter-channel delays were approximately 22.765 μs, suggesting slower internal clocks than the typical 5 MHz analogue to digital converter clock source that produced 21.8 μs delays between channels. FTSP timestamps enabled data from different motes to be synchronised in time, with a resolution of 30.5 μs. The tests showed timestamps on motes to be typically within 1 to 2 clock cycles of each other. Phase comparisons between channels on different motes can be performed, but with limited accuracy due to these offsets.

1.4 Limits and purpose

This research was limited to working with a single hop network and, although a range of sensors could be interfaced with the sensor motes, only accelerometers were used. A sample application is presented to illustrate a proposed use of the system, but interpretation of the results obtained from testing under the application conditions was outside the scope of the project.

This work forms part of a larger project to develop a condition monitoring system for power transformers.

1.5 Outline

A summary of the thesis outline is shown in Figure 1.3. There are 3 logical parts to the body of the thesis.

Chapter 2 discusses the available mote hardware and software. The chosen hardware platform—the TelosB mote—and software—TinyOS—are described. Chapter 3 presents related sensornet protocols, while Chapter 4 compares prominent time synchronisation protocols and justifies the decision for using the flooding time synchronisation protocol.

Chapter 5 describes some of the preliminary designs for the data collection sensornet, in order to illustrate the incremental design, while Chapter 6 details the current design, including mote and computer software and sensor selection.

The sensornet protocol is analysed and results of a simulation compared with empirical results in Chapter 7. Chapter 8 displays and discusses the results of testing the base system, i.e. the system without sensors attached. Results from tests using sensors in a controlled laboratory environment, and an application of the system on an isolation

transformer, are provided in Chapter 9.

Finally, the document is concluded and future work is considered.

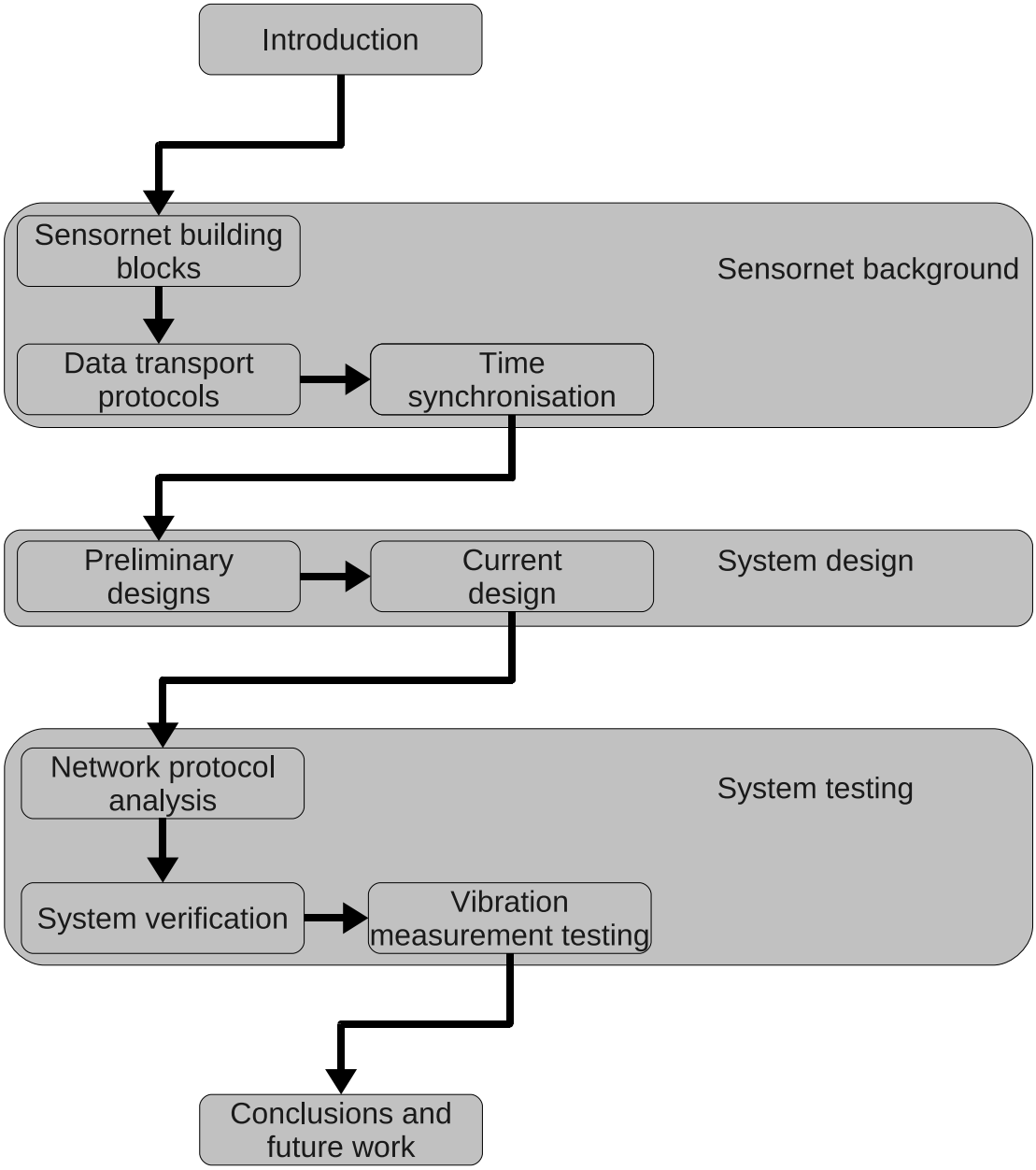


Figure 1.3: Outline of thesis.

Chapter 2

Sensornet building blocks

2.1 Mote hardware

There are a number of sensornet platforms available based on 8-bit micro-controllers [53, 61] through to 32-bit micro-controllers [60]. A widely used mote platform is the Telos revision B (TelosB) mote. The Telos mote was designed at the University of California, Berkeley (UCB) [70] and has been manufactured by a number of companies. One of the newer variants of the TelosB mote is the Epic core, designed by Dutta while at UCB [20]. The Epic core is not a complete mote but only contains the core components, allowing for rapid development of custom mote hardware around a predeveloped core. The Epic core platform is supported by the TinyOS operating system, facilitating easy software development.

It was decided that it would be beneficial to use an available platform rather than developing a new one, as it would already be tested, have software available and have community or company support. The TelosB platform was chosen as described below.

2.1.1 TelosB mote

The TelosB mote is similar to other 8-bit and 16-bit micro-controller based sensornet platforms. Its key advantages are the ultra low power MSP430 micro-controller [82] and its open design. The schematics and PCB design are available for download [86]. The mote is supported by multiple sensornet operating systems [18, 84]. In addition, the author had access to a number of TelosB motes and experience in working with these motes. The author also had access to Epic cores which could be used for customising hardware for future work.

A block diagram of the main components of a standard TelosB mote, such as those produced by Crossbow, can be seen in Figure 2.1. Crossbow's wireless sensor network product lines now belong to Memsic [62].

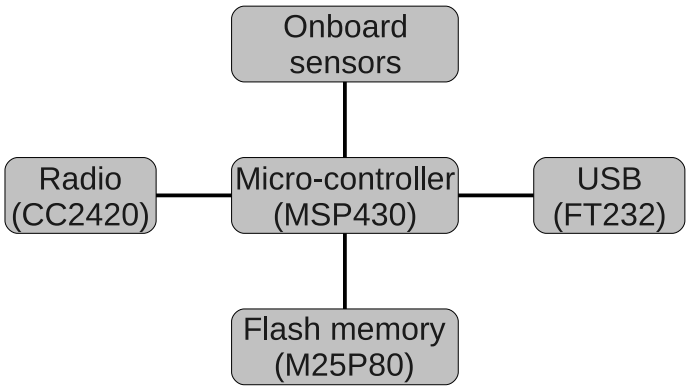


Figure 2.1: Block diagram of the basic TelosB mote components.

The two main components of the TelosB mote are the MSP430 low power micro-controller and the CC2420 2.4 GHz, IEEE 802.15.4 compliant radio [81] both from Texas Instruments. The MSP430 can be clocked at 8 MHz, has 48 kB Flash memory and 10 kB RAM, and has an integrated analogue to digital converter (ADC) [82]. The mote has a 1 MB external Flash memory IC and a selection of onboard sensors. The CC2420 radio IC is compliant with the 2003 version of the IEEE 802.15.4 standard [36]. The later 2006 version is backwards compatible with the older standard [37].

The standard TelosB mote has a USB connector for communicating with a computer, for programming the mote and for getting power from a computer. The USB functionality is provided by an FT232BM serial to USB converter [85].

Headers on the mote expose pins for the developer to use to interface the mote to external hardware. Six ADC channels are available from the header.

Analogue to digital converter

The ADC on the mote is integrated into the MSP430F1611 micro-controller [82] and has 12-bit resolution. The ADC is highly configurable with 8 channels, of which 6 are connected to headers on the TelosB mote and are available externally. The ADC can be clocked from multiple clock sources, including an internal 5 MHz oscillator. Sixteen memory registers are associated with the ADC, allowing 16 consecutive samples to be taken under the ADC subsystem's control and facilitating different conversion modes.

The ADC can operate in 1 of 4 conversion modes [80]:

Single-channel single-conversion — A single sample is obtained from a single channel.

Sequence-of-channels — Up to 16 samples across multiple channels can be converted.

Repeat-single-channel — A single channel is sampled repeatedly, with the sampled values stored in the same memory register. The memory must be read after each sample is complete so as not to lose data.

Repeat-sequence-of-channels — The *sequence-of-channels* mode gets repeated. The samples must be read after each sequence to avoid data loss.

For all modes, except *single-channel single-conversion*, multiple samples can be taken in rapid succession by setting the *multiple sample and convert* (MSC) bit (while in pulse sample mode) [80]. Only the first sample in a sequence needs to be triggered from a clock source when the MSC bit is set. The remaining samples are taken as soon the the previous sample and conversion has completed.

The channels are therefore sampled independently, with only one channel sampled at a time. As such there will always be a delay between samples, but by using the MSC bit this can be kept as small as possible.

2.2 Mote software

There are many software options for programming mote hardware. These options tend to depend on the hardware platform, although some platforms have multiple options for software.

Some hardware platforms are programmed in C using libraries made available by the hardware manufacturers. Sun Microsystems provides the Sun SPOT platform that is programmed in Java [79].

Several embedded operating systems are available for motes. While the Imote2 can run Linux [60], most motes require more modest operating systems.

TinyOS [84] and Contiki [18] are both widely used sensornet operating systems [17]. They are both open source operating systems, and have taken different approaches to dealing with the constraints of sensornets. The most noticeable differences between the two operating systems are that Contiki is programmed in C, implements threads and uses dynamic memory allocation [19], whereas TinyOS is developed in nesC and

has chosen to avoid threads and dynamic memory allocation [48]. However, threads are now available in TinyOS through the TOSThreads library [43, 44].

TinyOS was chosen for this project as the author had previous experience with the operating system.

2.2.1 TinyOS

TinyOS is an open source operating system for networked embedded devices. It is designed to operate on very limited devices. TinyOS is written in nesC, an extension of the C programming language, and was originally developed at UCB [32]. TinyOS is an event driven operating system. It allows for modular code that encourages code reuse and customisation.

TinyOS introduces 2 types of source code files. These are components and interfaces [47]. Components are further split into modules and configurations. Modules contain the program logic and state, while configurations list the connections between components.

All program states in a component are static, i.e. have local scope, and communication between components is done via the interfaces. Each component has a signature that gives the component's name, type (module or configuration), and the interfaces associated with the component [28, 47]. Components are connected to each other through common interfaces. These connections are defined in the configuration components. The connecting of components through interfaces is known as wiring. This modularity and wiring allows components to be easily replaced to provide differing functionality, and to create abstractions from the hardware to enable porting to different platforms.

TinyOS has 3 levels of hardware abstraction [30], the hardware presentation layer (HPL), the hardware adaptation layer (HAL) and the hardware interface layer (HIL) as seen in Figure 2.2. The HPL provides a nesC abstraction of the underlying hardware to present the hardware capabilities while hiding hardware complexities. The HAL builds on HPL components to provide platform-dependent functions like configuring and reading from an ADC channel. HIL components are further abstracted to provide platform-independent functionality. The HIL provides portability but is less efficient and will not necessarily implement all the functionality provided by the HAL. The example components in Figure 2.2 are all part of the ADC code and show the typical naming convention used. HPL files are prepended with Hpl, HAL files begin with the specific hardware name, and HIL files are named independently of specific platform hardware.

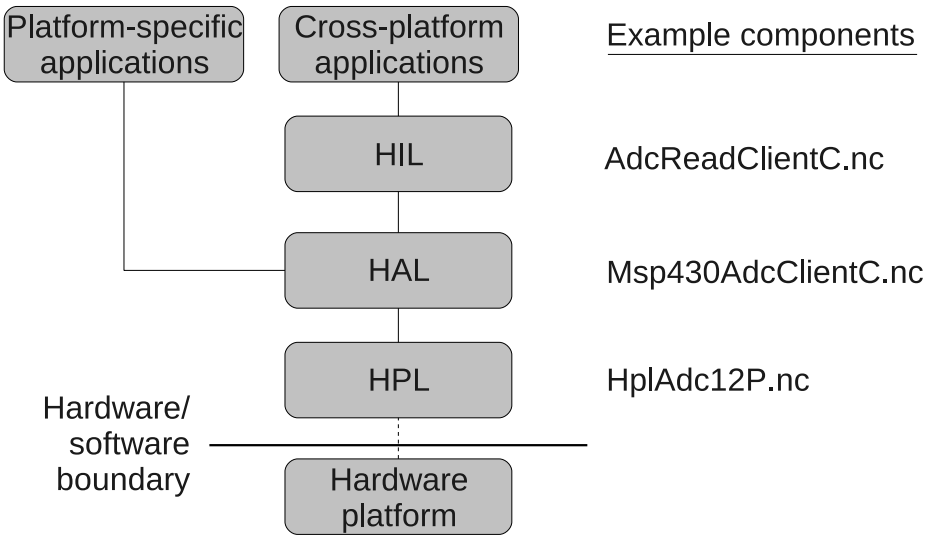


Figure 2.2: The three levels of TinyOS hardware abstraction. Modified from TEP 2 [30].

TinyOS provides a scheduler for tasks. Tasks are code blocks that the scheduler manages. The default scheduler uses a first-in, first-out (FIFO) algorithm that is used in this project. Tasks are non-preemptive with respect to each other. Each task is allowed only one spot in the scheduler's queue, which simplifies the queue while increasing reliability by ensuring that each separate task will be able to be scheduled [47, 48].

The TinyOS scheduler will put the micro-controller in sleep mode when there are no tasks to execute [47]. The micro-controller will wake on an interrupt. Interrupt code is referred to as async code in TinyOS. Async code can preempt other code, like tasks, so atomic blocks—blocks of code during which interrupts are disabled—may be required to ensure blocks of code run to completion uninterrupted.

TinyOS introduces a concept called split-phase [47] where relatively long running operations get split into separate requests and completion notifications. This is to prevent the micro-controller from being blocked while waiting for a response, typically from hardware, e.g. an interrupt to signal the completion of ADC sampling.

2.2.2 TinyOS tools and sample applications

TinyOS comes with a collection of tools for interacting with TinyOS applications, in addition to the supplied TinyOS source code. These support tools are available in C, C++, Java and Python. Not all of these languages have a complete set of tools. The C and Java tools were used in this project.

One of the primary tools is the *serial forwarder* program. This tool enables bi-directional communication between a mote attached to a computer and one or more computer programs. The serial forwarder provides a connection between a computer serial port or transmission control protocol (TCP) port and another TCP port. This enables communication between a mote and other applications. Only a single connection can be made to a serial port, but because the serial forwarder enables TCP connections, multiple connections can be made through the serial forwarder to a mote attached to a serial port.

A program called *serial forwarder listen*, sflisten in C, can be used to view the output being sent by a mote to the computer. The program connects to the TCP port of a serial forwarder program and outputs packets it receives to the standard output. The C version was used as the base for developing the data logging program. The Java version, Listen, was modified to output timestamps for each packet received to assist in debugging.

TinyOS includes a selection of sample TinyOS applications and some accompanying Java programs. Two applications were of relevance to this work: the oscilloscope application and the base station application. Oscilloscope is an application that samples a sensor periodically, storing the sampled values in a data packet. A packet is sent on the radio once the data packet's payload is full. Base station is an application that creates a bridge between motes and a computer. Base station will forward packets received from the radio stack to the serial stack and vice-versa. There are 2 buffers for transiting packets, one for each direction of transfer. Packets are dropped if a buffer is full.

Along with the sample applications are test programs to confirm aspects of TinyOS operation. One such set of programs tests serial communication. There is a TinyOS program to receive packets from the serial stack and display a count value on the LEDs. The program will periodically send a packet to the serial stack. The accompanying Java program will periodically send packets to the serial port to which the mote is attached and output the packet count from received packets to the standard output. The *serial test* Java program provides a convenient starting point from which to create programs for communicating from the computer command line to motes.

2.3 Summary

This chapter discussed sensornet hardware and software. The TelosB platform is an open design. It has a low power MSP430 micro-controller with integrated 8-channel

ADC and was chosen for development. The TelosB motes were programmed using the TinyOS operating system, a widely used sensornet operating system with an active community. A description of TinyOS and its support tools was provided.

Chapter 3

Data Transport protocols

This chapter presents a number of sensornet data transport protocols. The protocols can be described in terms of the direction of the data flows, the degree of reliability provided, application of congestion control, distributed versus centralised control, and rate of data transfer. The motivation for the development of our proposed protocol is presented after the related work has been discussed.

3.1 Collection Tree Protocol

The Collection Tree Protocol (CTP) [23] is a data collection protocol that incorporates routing and data transmission. A routing tree is created and regularly updated to provide paths to one or more data sinks. CTP is an anycast protocol, that is data packets will make their way to a data sink, but not necessarily one specific sink. As an aside, IPv6 added anycast addressing in addition to unicast and multicast [16]. CTP does not directly provide for reliable data transfer, but makes use of data link layer acknowledgements for data packets. CTP was designed for low data rate applications. The use of acknowledgements for data packets make it inappropriate for high rate data networks, adding to channel congestion and consuming additional energy.

Ganeriwal et al. created an implementation of CTP called CTP Noe [29]. The implementation was evaluated against MultihopLQI [90], a widely used collection protocol similar to CTP but designed for Chipcon CC2420 radios. The low data rate design is evident as CTP Noe supports aggregate data rates of approximately 30 packets per minute, while maintaining a low duty cycle of $< 3\%$.

3.2 DIsemination Protocol and Drip

The DIsemination Protocol (DIP) [54, 51] and Drip [83, 51] are both dissemination protocols, which are the reverse of the collection protocols described above. In dissemination protocols, data are transferred from one or a few nodes (data sinks) to many nodes (sensor nodes). DIP is more scalable than Drip, though more complex. Both are limited to small data payloads.

3.3 Deluge

Deluge [35] is a dissemination protocol like those above, but designed for reliably propagating large blocks of data. It's primary use is for reprogramming sensor nodes. Deluge is based on the Trickle algorithm [50] and can reliably transfer data through a large network relatively quickly, at 90 bytes per second.

3.4 Sensor Protocols for Information via Negotiation

Heinzelman et al.[31] present two adaptive data dissemination protocols known as Sensor Protocols for Information via Negotiation (SPIN). These protocols are resource aware and data-centric, in the sense that they attempt to conserve energy by negotiating with nodes to prevent multiple nodes from sending data that results in redundancies.

3.5 Event-to-Sink Reliable Transport

The Event-to-Sink Reliable Transport (ESRT) protocol [2] was designed to reliably transport event information from the area of event detection to a data sink. It was designed to cater for sensor networks where notification of an event is more important than delivery of individual messages. It therefore does not provide reliable end-to-end delivery for nodes.

3.6 Pump Slowly, Fetch Quickly

Pump Slowly, Fetch Quickly (PSFQ) [87] is a reliable transport protocol for sensor networks. Packets are transmitted relatively slowly, allowing hop nodes along the route to make local data recovery before the next packet arrives. PSFQ makes use of negative acknowledgements (NACK). Only once a receiver detects that packets are missing, i.e. receives a packet with a sequence number higher than expected, does it send a NACK requesting the missing packets. Each node along the route, therefore needs to keep a subset of the packets it sends in memory in case packet recovery is required.

3.7 Reliable Multi-Segment Transport

Reliable Multi-Segment Transport [76] is a reliable transport protocol designed to operate on a network using Directed Diffusion [38], a data-centric communication protocol–nodes are addressed based on data of interest rather than having explicit network addresses. RMST makes use of data link layer and transport layer operations. It advocates the use of automatic repeat requests (ARQ) at the data link layer for unicast transmissions, while, like PSFQ, makes use of NACKs at the transport layer. Two versions are presented, a caching version where recovery is performed en route, or the default end-to-end recovery version relying on ARQs for a measure of single hop reliability.

3.8 COngestion Detection and Avoidance

Wan et al. present COngestion Detection and Avoidance (CODA), a system for congestion control in sensor networks [88]. CODA provides both distributed and centralised congestion control. CODA works in a distributed fashion, with receivers detecting congestion and notifying upstream (in the direction of the source) nodes of this congestion. Once a threshold has been reached, the data sink will take control of managing congestion by issuing acknowledgements to received packets to control the rate of flow. CODA does not explicitly provide reliable delivery, however control of network congestion should enable fairly reliable transfer, even without the addition of reliability mechanisms.

3.9 Interference-aware fair rate control

Interference-aware fair rate control (IFRC) [71] is a distributed algorithm for managing congestion fairly and efficiently. IFRC has three mechanisms for dealing with congestion. IFRC monitors the average queue length, for each node, of packets generated by the node and received by it for forwarding. IFRC assumes that a tree-based routing protocol is used, preferring a protocol using link quality as a metric, such as MultihopLQI. IFRC includes congestion information in each packet sent, so that nodes may inform others of their – and their child nodes' congestion levels. The mechanism introduces 2 constraints: a node may not transmit at a rate higher than it's parent node, and a node must set its transmission rate to the lower rate of a neighbour node, or the neighbour's child node. Neighbours are nodes that are in radio range, but not necessarily in the same branch of a routing tree. These constraints enable nodes in interference range to reduce their data rates to minimise congestion. Nodes implementing IFRC generally increase their data rates additively. A node will halve its rate once it has exceeded a congestion threshold, and will reduce its rate according to the 2 constraints above. To aid start up and recovery from heavy congestion, a node will implement multiplicative increase. The data sink also maintains a data rate based on what it receives, and can adjust and communicate this rate to the connected sensor nodes.

3.10 Rate-controlled reliable transport

The rate-controlled reliable transport (RCRT) [69] was designed for loss-intolerant, high data rate networks. It takes into account the limitations of sensing nodes by placing most of the complexity at the data sinks, which are assumed to have more resources available to them. RCRT provides end-to-end reliability through the use of NACKs. Data sinks determine the congestion level of network. Congestion determination is based on the time it takes to recover lost packets. Low losses are ignored by the congestion control scheme and simply handled by NACKs. The congestion scheme only seeks to drop transmission rates once a threshold has been reached. RCRT uses additive increase, multiplicative decrease to get an aggregate data rate across the network. To achieve this, individual flow rates are adjusted according to some predefined policy. RCRT is designed to allow multiple applications to operate across the network. This policy allows different applications, with different needs, to have their flow rates adjusted accordingly. RCRT leaves the routing to separate routing protocols. The TinyOS MultihopLQI algorithm was used in their empirical tests.

3.11 Wireless reliable acquisition protocol

The Wireless Reliable Acquisition Protocol (WRAP) is described by Liang et al. as part of their RACnet sensornet [52]. RACnet was designed to monitor the environmental conditions of data centres with the aim of reducing cooling costs and thereby power consumption. WRAP uses a number of mechanisms to reduce congestion in the dense network, provide reliable data collection and provide global timestamping. WRAP generates routing trees in a distributed manner and to reduce congestion has routing trees operating on different radio channels. WRAP uses token passing at the transport layer to further avoid congestion. The data sink generates a token and passes it along a routing tree to collect data. Motes attempt to calculate the time it takes for packets to propagate through a tree to adjust the delay between transmissions. Data sinks make use of NACKs to provide end-to-end reliable data collection, and flooding time synchronisation protocol (FTSP) mechanisms for distributing a global clock for timestamping data. WRAP typically collects greater than 99 % of node data.

3.12 Motivation for proposed protocol

The protocol presented in this thesis was designed for a specific application, rather than as a general-use protocol like most of the protocols described above. Though the proposed protocol could be adapted for other uses if required. The application was to collect large amounts of synchronised time series data from multiple points on a high power transformer. It was additionally requested that the data be logged rapidly to enable frequent measurements during the testing phase. The application was required to sample data at a high sample rate. Though high sample rates are typically required for structural monitoring applications [69], high sample rates are still a rare requirement in sensornet protocols. The sample rate presented limitations of the sensornet platform. It was not possible to collect timestamps during sampling periods without missing timer interrupts. Along with the high sample rate, many consecutive samples were required for time series analysis and conversion to the frequency domain.

Multiple sensors needed to be compared to each other in the time domain, requiring timestamping and synchronised data capture. The requirement that the sensornet be placed on a single power transformer, led to the decision to use a single hop network. The number of nodes that would be required on a transformer would limit the size of the network.

These factors led to a protocol that had a 'batch' operation. The protocol would put the entire network into specific states, e.g. data collection. Having a batch operation with known, expected numbers of data packets, and limited sensor motes, meant that there was no need to provide data recovery during data collection. This had the benefits of simplifying the operation and being more efficient than standard NACK based data recovery, though the mechanism for detecting missing packets is the same as would be used in a NACK algorithm. With a NACK algorithm, a recovery packet would be sent each each time the base mote detected a gap in sequence numbers. If missing packets were not congruous, then a recovery packet would be required for each lost packet. Our proposed protocol is not affected by non-congruous losses as all lost packets are combined and requested in bulk, limiting the number of recovery packets to be sent. This method does require that the base station have sufficient RAM to log all missing packets. This did not prove to be a problem. It is a common assumption that the base mote will have more resources than sensor motes. RCRT specifically states this assumption, while all the protocols that implement centralised control have base motes with more complexity than the sensor motes.

The vast majority of packet transfer is from the sensor motes to the base mote, making this predominantly a collection protocol. The single hop network, and batch operation eliminates the need for a dissemination algorithm as during packet dissemination, only the base mote is using the channel and no forwarding of packets by sensor motes is needed.

Unlike many of the protocols above, the proposed protocol only deals with network congestion passively by staggering transmissions to lessen the chance of collisions on the air. This, again, simplifies operation and leaves recovery to the end-to-end recovery mechanisms in the recovery stage.

3.13 Summary

A selection of protocols from the literature were described. These protocols have varying designs, but typically cater for larger, more varied sensornets with lower data rate requirements. The application driving the development of the proposed sensornet had specific requirements that our design centred on, resulting in a protocol that had a batch-type operation.

Chapter 4

Time synchronisation

For many applications it is important that code on different nodes runs at the same time or is known to occur at a time relative to a common time, within some margin of error. This becomes more difficult if dispersed nodes need to be synchronised, and even more so if the communication channel has varying delays or is unreliable.

As with all aspects of sensornet development, certain criteria must be evaluated to determine the suitability of use. These include memory constraints, processing constraints, amount of data transmission, and transmission limitations. These constraints are all due to limited hardware resources and the need to keep energy usage down to maintain the long operating lifetimes expected of sensornets.

4.1 Radio transmission delays

Understanding the delays in radio transmission assist in developing and evaluating synchronisation schemes. Sensornet operating systems typically enable low level access to hardware, making it possible to design protocols that eliminate some of the delays.

Six delay elements have been identified in the transmission of radio packets. These are shown in papers by Maroti et al. [56] and Ganeriwal et al. [24]. The 6 elements are:

Send time — This is the time taken to build a packet and have it sent to the medium access control (MAC) layer.

Access time — The time taken from reception of the packet by the MAC layer

to gaining access to the physical medium. IEEE 802.15.4 uses carrier sense multiple access with collision avoidance (CSMA-CA) and will need to check that the channel is clear before transmitting.

Transmission time — This is the time it takes for the transmitting node to send a packet. It is dependent on the size of the packet and the data rate of the radio. Ganeriwal et al. [24] note that there may be small variations in the transmission time due to software interrupt response times. This should not be an issue for the TelosB platform as the radio IC has buffer space for a full physical layer protocol data unit, unlike the radio IC on the Mica2 platform used by Maroti et al. [56] and Ganeriwal et al. [24] for testing. The transmission time for a TelosB mote will be the size of the packet in bits divided by 250 kbps.

Propagation time — This is the actual time it takes for the data to propagate over the air from the sender to the receiver. This value is very small.

Reception time — This is the time it takes for a packet to be received by the radio and be passed to the MAC layer, and is similar to the transmission time.

Receive time — This is the time it takes for the MAC layer to get the packet to the application for use.

4.2 Network time protocol

Probably the most widely used time synchronisation protocol is the network time protocol (NTP) used on the Internet [64]. This document refers to NTP version 3 [65] as version 4 [64] was only published as an RFC in June 2010.

NTP [65] allows for connections between NTP hosts in different modes: symmetrically, client/server or broadcast. Nodes communicating in symmetric mode are all able to provide and request timing information, while in client/server mode the client will request timing data from the server. Broadcast mode was designed for a single NTP node to supply timing information to many nodes on a high speed LAN. Typically 2 packets will be involved in each synchronisation packet exchange, unless the broadcast mode is used. Only the NTP server will send NTP packets in broadcast mode.

Broadcast mode would be the most suitable mode of operation in a sensornet from an energy conservation perspective . However, broadcast mode provides less accuracy than the other modes [65].

The packet defined for NTP is shown in Figure 4.1 and has a size of at least 384 bytes, making it too big to be transmitted in a single IEEE 802.15.4 data frame [37].

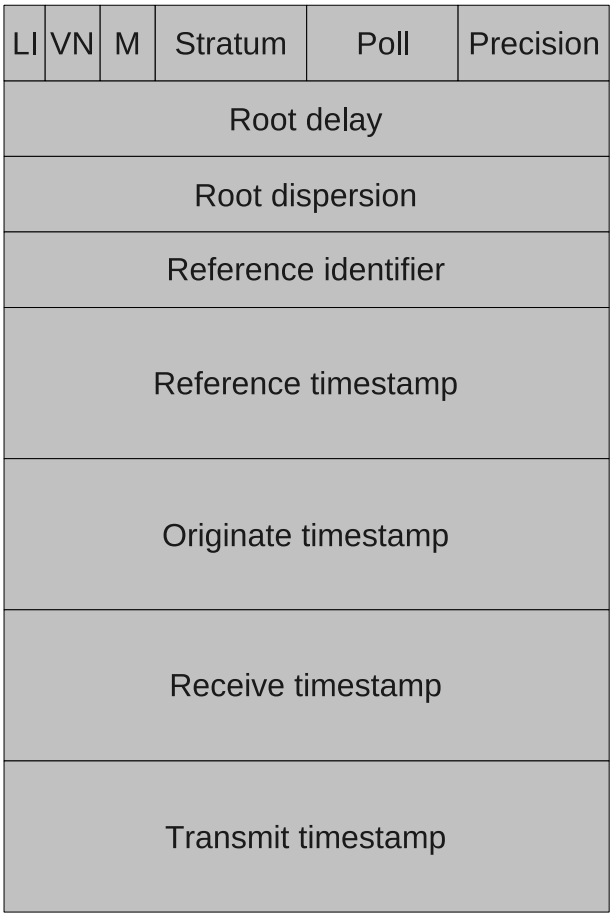


Figure 4.1: The NTP packet format. Not shown is the optional 96-bit authenticator field after the transmit timestamp. Modified from Figure 4 in RFC 1305 [65].

NTP is a fairly complex protocol, but a simplified version of NTP known as simple NTP (SNTP) is available and uses the same packet format as NTP [63]. NTP was not designed specifically for wireless networks and as such the RFC makes no mention of radio delays, but simply states that careful design is necessary to maintain the accuracy of the servers [65].

4.3 Reference-Broadcast Synchronisation

The Reference-Broadcast Synchronisation (RBS) algorithm [21] was designed to work over sensornets or other broadcast networks. RBS works by nodes generating timestamps on reception of broadcast packets. The broadcast packets need not contain any timing information and so any broadcast message can be used. Receivers communicate their timing information with each other. This approach eliminates the variable

delays of the send time and access time. The propagation time is very small so each node should receive the broadcast packet almost immediately.

Although only a few (possibly 1) broadcast packets are sent, each receiving node will share timing information. The number of time synchronisation packets sent becomes significant.

4.4 Timing-sync Protocol for Sensor Networks

The Timing-sync Protocol for Sensor Networks (TPSN) [24] is a time synchronisation protocol that provides a reference clock across a sensornet. It is a flexible protocol and can be adapted to suit different sensornet requirements.

TPSN uses a hierarchy of nodes, a spanning tree, and designates the reference clock node to be the root of the tree. The root is level 0 in the hierarchy. Each level in the hierarchy are neighbours of the previous level's nodes, that is those nodes in direct radio range of the lower level nodes.

The protocol is initiated by the root node. The root node will send a level discovery packet with its node ID and level number. Receiving nodes will repeat this activity using their node ID and level number in the packet they send. This is known as the level discovery phase. A node will send a level discovery request packet if it times out without having received a level discovery packet. It will join the tree at the lowest appropriate hierarchy based on the responses it received.

Once complete, the protocol transitions to the synchronisation phase by the root node sending a time sync packet. Level 1 nodes will respond by sending a sync pulse packet containing the time the packet was sent. The level 0 node will acknowledge each sync pulse packet with a packet containing the time the sync pulse packet was sent, the time it was received and the time the acknowledge packet was sent. Level 2 nodes will overhear the level 1 sync pulse and send their own sync pulse to a level 1 node that will respond with an acknowledge packet. This will continue throughout the network.

It can be seen from the TinyOS implementation of TPSN that the time sync packet and sync pulse packet are the same, and the sync packets are sent from the lower node to the higher level node [45].

TPSN introduces timestamping at the MAC layer, thereby eliminating the send and access times from the sender delays and the receive time at the receiver [24].

Pairwise synchronisation, however, results in a significant number of time synchronisation packets being sent in the sensornet.

4.5 Flooding time synchronisation protocol

The Flooding time synchronisation protocol (FTSP) [56] was designed specifically for wireless communication and the constraints of sensornets. FTSP has been designed to scale up to large networks and to recover quickly from node and network problems.

FTSP makes use of broadcast packets rather than the 2 packet exchange used by other protocols. FTSP uses MAC layer timestamps to reduce the inaccuracies introduced by the transmission delays. All delays are removed or compensated for except the propagation delay, which is very small. Linear regression is used by the receiving nodes to determine their offset and skew relative to the reference clock.

A simple root election protocol is implemented to select the root on start up and if the root becomes unavailable.

FTSP was chosen for its relatively high accuracy and suitability for sensornet applications. FTSP has been tested in real life applications, and libraries to implement the protocol were included with the TinyOS distribution.

FTSP has been used, sometimes in a modified form, for volcano and structural health monitoring. Sensornets were deployed at two volcanoes in Ecuador [90, 91] and at Mount St. Helens [75]. An experiment in structural health monitoring was performed on a scale model of a 4 story building by Chintalapudi et al. [12] and a deployment was tested on the Golden Gate Bridge [41]. FTSP was also used in a scheme to uniquely identify sensornet nodes based on the node's clock skew [34]. The paper introducing FTSP mentions the protocol's use as part of a countersniper application [56].

4.6 Other clock synchronisation protocols

Hong and No [33] propose a time synchronisation protocol similar to TPSN, in that it first has a spanning tree building phase and then synchronisation is initiated by the root node sending a sync packet. This protocol does not use explicit acknowledgement packets but makes use of the broadcast nature of the wireless channel. A parent node will overhear a child node sending a sync packet to a grandchild node and use this

packet as an acknowledge packet. MAC layer timestamping is used.

Lee and Choi [46] propose the Chaining Clock Synchronisation (CCS) scheme. CCS uses a subset of sensornet nodes to keep all network nodes synchronised to a reference node. The subset of nodes are chosen such that their collective transmission ranges cover the entire sensornet. This conserves energy by limiting synchronisation data transmissions. Clock synchronisation is done by having a 2 packet exchange between a sender and receiver. The packet count is reduced by combining 2 packets into 1, such that the node sending the reply is sending the request to the next node at the same time. CCS propagates clock skew data to enable nodes further from the reference node to calculate their clock skew relative to the reference node. This lessens clock error through the network.

Chaudhari and Serpedin [11] propose using a quadratic model rather than a linear one for modelling the relation between clocks, using a two message transmission scheme as used in many previous protocols. Their paper states that a single message scheme could be used with minor modifications. The authors state that the quadratic model is more accurate and allows for time synchronisation over extended periods of time.

4.7 Summary

Sources of delays in radio transmissions have been listed and a selection of time synchronisation protocols have been described, including how each protocol deals with the delays. Packet size and the number of packets sent are considered, as using the radio consumes a significant amount of energy. FTSP was chosen for this project as it requires low power, has relatively high accuracy, has been used in a number of deployments and is available with TinyOS.

Chapter 5

Preliminary sensornet designs

5.1 Overview

This chapter describes the development of the sensornet data collection system from the preliminary designs leading to the current design shown in the next chapter. The general form of the system is shown in Figure 5.1. It consists of three parts: the sensor motes, the base mote and the logging software. It is indicated in the text where versions differ from this form.

The design was developed incrementally and began with the TinyOS Oscilloscope application on a sensor mote and the base station application on the base mote. Prior to the current design, the focus had been on developing sensor mote software while keeping the standard base station application as the interface between the sensornet and the computer. Having prebuilt software simplified and sped up development.

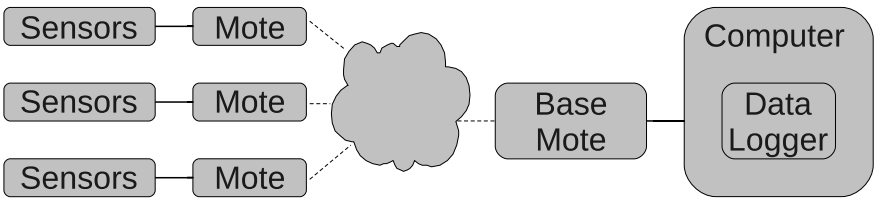


Figure 5.1: Overview of complete system.

5.2 Initial experimentation

The first version of the system was used to test various accelerometers and do early measurements on an isolation transformer. The single sensor mote used the oscilloscope application, modified to complete a sample session before transmitting the data. Sampling would begin automatically, 5 seconds after start up to avoid any influence from the user while starting or resetting the mote. The ADC channel was sampled periodically with the sampled data stored in a large buffer. The ADC code was changed from platform independent (HIL) components and interfaces to lower level hardware specific (HAL) code to improve efficiency. Packets were only sent to the base station mote once the buffer was full and sampling had been completed.

This program was then modified to begin a sample session on receipt of a control packet rather than physically having to restart the sensor mote. The system then consisted of 3 motes. These were the sensor mote and 2 motes attached to a computer, a sender mote and a receiver mote. The user would send control packets through the sender mote using a modified version of the TestSerial Java program. Data packets from the sensor mote were logged from the receiver mote using a modified version of the Listen Java program.

5.3 Initial multi-channel work

After the initial work, the program was modified to read from multiple ADC channels. These channels could be read either:

sequentially, where each requested channel was sampled until the buffer was full and the data was transmitted before the next channel sampled, or

in parallel, where all requested channels were sampled each sample period.

The operation was as follows. The sensor motes would wait until they received a control packet from the user. The control packet used bitmaps for selecting which channels were to be sampled. The user could select either sequential or parallel operation. The program would request access to the ADC and once granted would configure the channels and begin sampling. The data was transmitted once a sample session was complete and the mote would return to waiting for a control packet. In the case of a sequential read, the sample session and data transmission would be repeated as necessary before waiting for the next control packet. It should be noted that

TinyOS puts the micro-controller in a low power sleep mode while inactive (waiting for a new packet to arrive).

5.4 Multi-channel parallel only operation

The focus of the work was on the parallel operation, as we were interested in comparing the values from different sensors in time. As such the program was split into a parallel only version and a sequential only version. These could be merged later. This split allowed for simplification of the program and for focus to remain only on the operation mode of interest.

The system consisted of a sender mote and a receiver mote, both running the standard TinyOS base station application connected to a computer, and multiple sensor motes running the software described below and shown in Figure 5.2.

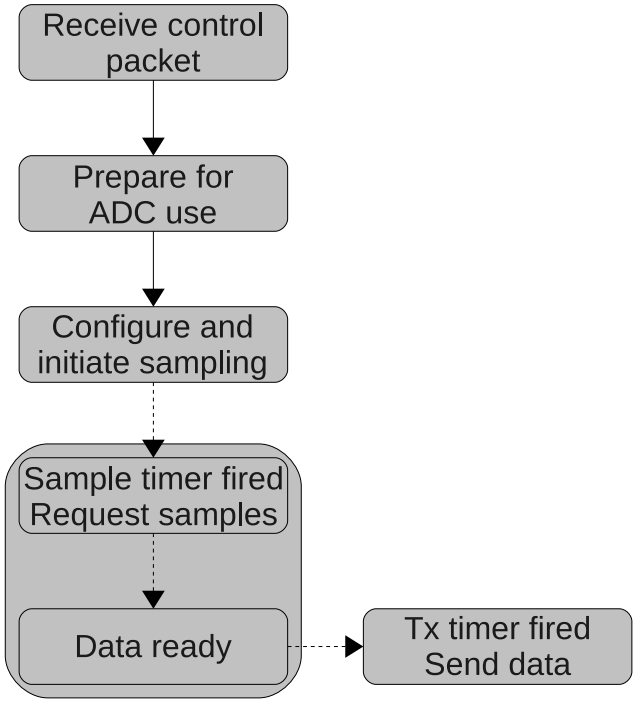


Figure 5.2: Diagram showing operation of sensor mote in preliminary software.

The sensor motes waited until reception of a control packet, indicating which sensors were to be sampled and the required sample period. Preparations to use the ADC were then made and access to the resource was requested from the operating system. The ADC was configured for multiple channel sampling, and a sample timer was set using the sample period obtained from the control packet once access to the resource had been granted. Samples were stored in a large queue. A multi-channel sample request was made each time the sample timer fired, provided the storage queue was not full.

The first few samples were discarded to eliminate spurious data. A transmission timer was then started to enable periodic sending of data to the computer while there was sufficient data in the queue for transmission.

The use of a queue and transmission of data packets during the sample session allowed for collection of more samples than could ordinarily be stored in memory, as samples were periodically removed from the queue. The rate of data entering the queue was always higher than that of data being removed from the queue to prevent perpetual sampling. Perpetual operation would not be desirable in a final product as it would consume too much energy.

Transmitting data packets during a sample session was not feasible as it would limit the maximum sampling rate. Once the last data packet had been sent, the microcontroller would return to a sleep mode until the next control packet arrived.

5.5 Automated multi-channel parallel operation

The system needed to be automated for a demonstration at the 7$^{\text{th}}$ ACM Conference on Embedded Networked Sensor Systems (SenSys 2009) [40]. Four sensor motes were required, each with 5 sensors attached. One mote had sensors on a cantilever, while the remaining 3 motes had sensors on a vibrating platform. The sensor mote software was modified to allow one of the sensor motes to become a master mote to facilitate the automation. The master mote performed initialisation and synchronisation of the motes. Only a single base station mote was required for this version. The sample period and number of sensors per mote were preset so the control packet was not used.

A new packet was introduced for network initialisation and synchronisation. This packet had 3 fields: a mote ID field, a hello field and a start time field. The ID field contained the mote ID of the mote sending the packet. The hello field was set by the master and modified by the slave in its reply. The start time field was used to indicate the delay from receiving the packet until sampling must begin. At start up, the master mote would periodically send this packet until it had received a response from all sensor motes in the network. It would then set the start time field and send this packet to the motes. The operation from here was as before, with a sample session being completed before the data was transmitted. It was decided to loop through this operation rather than checking that all motes were active before each sample session. This behaviour was changed in a later version.

A problem with the master/slave program was a noticeable offset in sample times between the master and the slaves, which appeared to be closely synchronised. The master mote had to have a small delay added to compensate for the time taken by the slave motes to receive the packet. This delay had to cover all six radio transmission delay elements described in chapter 4. The delay was fairly consistent as the channel was always available at the time the master sent the control packet to initiate sampling, essentially eliminating the access time. The slave motes should have near identical timing as they run identical software. Relative differences in starting times would produce offsets between motes.

Data was logged using a C program expanded from the sfListen program to log each mote's data to a separate comma separated values (CSV) file.

5.6 Summary

This chapter described the preliminary designs showing the incremental design process, with each program building on previous designs.

Chapter 6

Current sensornet design

The current design stems from the experiences gained from the previous work and the need to improve on the previous designs. The requirement was to create a simple, reliable, single hop wireless data collection system that could be used in the laboratory and on an isolation transformer on the CPUT campus. The decision was to keep the sensor motes uniform to avoid additional synchronisation issues, as before, and to keep them as simple as possible.

The system is shown in Figure 5.1. The sensor motes collect the data from their attached sensors on request and forward the data to the base mote. The base mote performs network management tasks, requests data from the sensor motes and sends the data to a program for logging through its physical connection to the logging computer. The base mote also supports reliable data delivery. The logging software stores data collected from the base mote and writes these data to timestamped files. The files can then be used in MATLAB for data analysis. The system operates using a finite state machine. The state transitions are controlled from the base mote and the state transition diagram is shown in Figure 6.1.

All communications take place by way of packets. Each defined packet has a different structure depending on its purpose.

6.1 Packet structures

Five different packets are defined and illustrated in Figures 6.2–6.6. The values above the packets in the figures are the sizes of the fields in bits.

The first packet, Figure 6.2, is the data packet. The data packet takes the same form

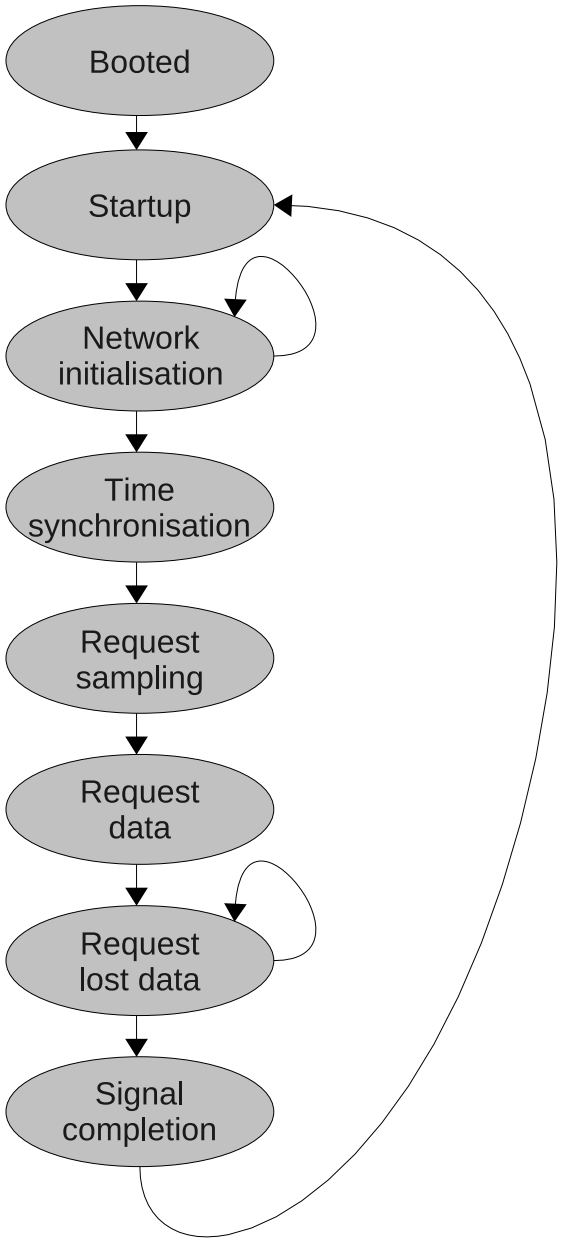


Figure 6.1: State transition diagram from the base mote.

as the packet from the TinyOS oscilloscope sample application [27]. The version field is not used. The interval field contains the sample period in milliseconds. The sample period is set by the base mote. The ID field is the 16 bit identifier of the sending mote. The count field is a packet number that is used for ordering and determining missing packets. The final field is for the accelerometer readings and is set for ten 16 bit values.

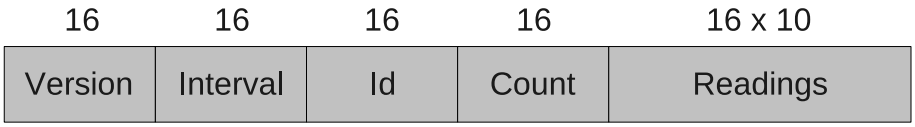


Figure 6.2: Structure of the data packet.

The remaining packets are all control and meta-data packets.

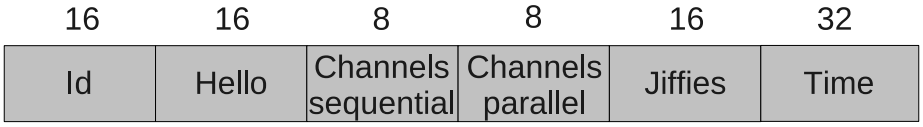


Figure 6.3: Structure of the primary control packet.

The primary control packet, Figure 6.3, has 6 fields starting with the sending mote's ID. The second field is used for network attachment. The base will set the field with a particular code, and each mote receiving this packet must respond with a separate code in this field. The next two fields are bitmaps for selecting which ADC channels are to be used for sequential and parallel channel sampling. The final two fields are the sampling period and the time to start sampling. The sampling period field gets its name from the clock units, which are informally referred to as jiffies.

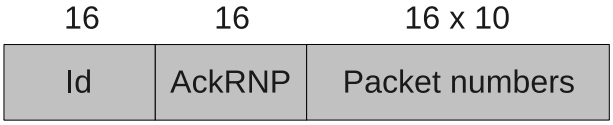


Figure 6.4: Structure of the control packet used for enabling reliable data packet collection.

The reliability packet, Figure 6.4, begins with the sender mote's ID and is followed by a field containing the number of packets that need to be resent. This field was chosen to have multiple purposes but is currently only used for the abovementioned purpose. The final field contains the packet numbers of the packets that need to be resent.

The timestamp packet, Figure 6.5, simply contains the ID of the sending mote and timestamps from that mote.

Figure 6.5: Structure of the timestamp packet.

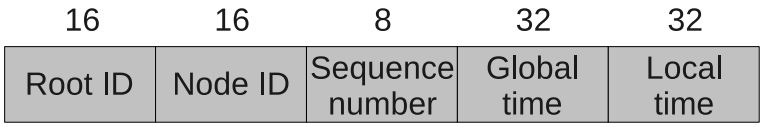


Figure 6.6: The structure of the FTSP time synchronisation packet.

The time synchronisation packet, shown in Figure 6.6, is specified in the TinyOS FTSP TimeSyncMsg.h file [58]. The packet contains the mote ID of the synchronisation root, the mote ID of the sender, a sequence number and global and local timestamps.

6.2 States

The system operates in a loop and shifts through 7 possible states (Figure 6.1) to enable the networked data collection. The 7 states are:

- Startup,
- Network initialisation,
- Time synchronisation,
- Request sampling,
- Request data,
- Request lost data, and
- Signal completion.

Transition between states is typically based on timers.

6.2.1 Startup

Startup is a simple state that is used to reset the program state. Transition from startup to network initialisation is through a short timer.

6.2.2 Network initialisation

The network initialisation state allows for the network to become ready to receive sampling requests. In the current implementation, ready is considered to be once all sensor motes have indicated to the base mote that they are active and prepared to begin the data collection process. The base mote will send a control packet periodically until each sensor mote has responded. The motes stagger their responses as a passive means of congestion management. Only two fields in the control packet are used for network initialisation: the ID field and the hello field. The ID field is set to the sending node's network ID as always. The hello field is set to 0x55 by the base. The sensing motes will set the hello field to 0xAA in response. All other fields are set to zero. The transition from the network initialisation state to the time synchronisation state will only occur once all sensing motes have responded to the hello request.

Previously the system was designed to use the standard TinyOS base station program as the interface between the radio (sensornet) and serial communication (computer). Therefore one of the sensor motes was designated as a master node and performed the network initialisation and sampling requests. This had the synchronisation issue, described in Section 5.5, where the master's samples were offset from the slaves' samples. An extra delay had to be introduced to the master to compensate for the offset. Reliable data collection and FTSP timestamping had not yet been implemented—this would mean having master and slave sensor nodes and still require changing from the standard TinyOS base station program. It was therefore decided to have all the sensor motes operate identically and shift all control to the new base mote software.

6.2.3 Time synchronisation

The time synchronisation in the system makes use of the TinyOS implementation of the flooding time synchronisation protocol. Time synchronisation can operate in one of 2 modes: the master clock source can periodically send time synchronisation packets, or the sending of packets can be controlled by the application [57]. The second option was chosen to gain control of time synchronisation packet transmission and to limit the synchronisation transmission to only when it is required. Motes compete to become the master clock source in the implementation, but only the base mote is programmed to become a master clock source. A mote will become a master clock source if it has a node ID lower than the other prospective clock sources, or it will default to being one if it has not received a time synchronisation packet within a certain number of time synchronisation transmissions [57]. The base mote therefore

sends sufficient packets for it to default to a clock source. The base then sends 10 time synchronisation packets at regular intervals to enable each sensor mote to synchronise with the base mote. A minimum of 8 timestamps are required by the receiving motes to consider themselves synchronised. Sending the extra packets minimises the chance that a mote might miss a packet and not have its clock synchronised before the next state transition.

6.2.4 Request sampling

The request for the sensor motes to prepare for sampling is sent by the base mote in this state. Data passed in the request includes a bitmap indicating which sensors to sample, the sample rate and the time from reception of the packet at which to start sampling. Global time was not used for the start time as it was decided to keep the operation as simple as possible. The sensor motes receive the packet at the same time. The propagation time should be equal and very small. The reception time should also be equal as this is handled by the same radio IC on each sensor mote. The largest source of delay would be in the receive time, where the packet is passed from the MAC layer to the application for use. The relative start time was shown to be adequate. However, a change to global time may be introduced in the future to enable expansion to a multi-hop network.

On reception each mote starts a timer using the value it received from the packet and configures the requested ADC channels in preparation for sampling to commence. The sensor motes also take a timestamp reading before sampling begins. Previously, timestamps were taken during the sampling session, but it was found that obtaining the timestamp had an effect on the sample period at the currently used sample rate of 1 kHz. Another timestamp is obtained after the sample session has completed.

The sensor motes make use of the micro-controller ADCs sequence-of-channels mode through the TinyOS Msp430Adc12MultiChannel interface. The ADC is configured to use the internal ADCs 5 MHz oscillator and the multiple sample and convert (MSC) bit is set to enable fast switching between channels. The sensor mote places the samples in a buffer. The first 20 samples collected are discarded to avoid any irregularities that may occur at the beginning of the sampling session. Data collection ends once the buffer is full at 3000 samples. A buffer of 3000 samples fills most of the available RAM on the micro-controller. The sensor mote software will insert a set of invalid values into the buffer if a request to sample from the ADC is denied during the sample session. This ensures that timing is maintained across motes.

6.2.5 Request data

The request for data is implicit and no control packets are sent to initiate this. This state is therefore just a timed wait from the base mote's perspective. Each sensor mote will send its collected samples once the sample set has been collected and a post collection timestamp has been taken. The data packets are sent at short intervals and contain 10 samples each. Offsets are used to stagger the transmission of the packets from different motes. The base mote passes each received packet to a buffer between the radio reception and serial transmission. It also keeps track of lost packets. Packets are received in order, making lost packets easy to detect. The serial link speed is 115200 bps, which is slightly less than half the radio transmission speed. This causes a bottleneck and packets may be dropped once the buffer becomes full. Packet numbers from packets lost over the air and dropped from the buffer are logged so that they can be recovered later.

The logging software keeps buffers for data and metadata for each mote. The data and metadata received from the base are stored in the relevant buffers during this state.

6.2.6 Request lost data

Laboratory experimentation showed very limited packet loss, prior to data recovery, and the energy expended in requesting many packets to be resent is high. It was therefore decided to set a threshold on the number of packets that would be recovered. Currently this is set to 30 packets per sensor mote–10% of the transmitted data packets. If more than the threshold number of packets have been detected as missing, they are ignored and the system transitions to the next state.

The code for recovering lost packets consists of 3 parts: the *management* part that calls the *sender* part, and the *receiver* part.

The management part begins by checking whether there are any outstanding packets for each mote. If all data was successfully received or if any mote had more outstanding packets than can be recovered then the base transitions to the next state.

The sender part builds and sends control packets containing the number of missing packets and their packet numbers to each mote with outstanding data packets. Control is returned to the management part once a packet for each mote with missing data has been sent.

The receiver part receives data packets from the sensor motes. If the resent packets are successfully sent to the serial stack, i.e. recovered, then their packet numbers are removed from the missing data list.

The state transition therefore only happens once the management part has determined that the transition can be made, because all the packets were recovered or the losses were too great to initiate the data recovery.

6.2.7 Signal completion

The base mote signals completion of a sample cycle to the logging program once data collection has finished and any lost packets have been recovered. On reception of this signal, which takes the form of a control packet, the logging program writes the data in its buffers to files. The current computer date and time is used for naming files. Each file, as part of a sample set, has the same computer time as part of its file name. All files are given a *txt* extension. The timestamp packets are stored 1 packet to a line in a file with the computer time as the file name. The packet number and data fields from each mote are stored as 2 columns in CSV files. Each mote's data has its own file with a letter appended to signify to which mote the data belongs. The letter *L* is appended to files with incomplete data sets.

Below is an example of the file names for data collected on the 12th of May 2010 just before 14:00 SAST:

```
2010-05-12T13:55:40+0200.txt
2010-05-12T13:55:40+0200-a.txt
2010-05-12T13:55:40+0200-b.txt
2010-05-12T13:55:40+0200-c.txt
```

The first two packets of data from mote A are shown below. It can be seen that the first column contains the packet numbers, while the second column contains the data fields. The data is output in the order it was received in the packet, which is the order that data was received from the ADC buffer:

```
0001,03a9
 ,03b3
 ,03b7
 ,0775
```

```
 ,0853
 ,078f
 ,078f
 ,0793
 ,083b
 ,08ab
0002,020f
 ,020e
 ,01f9
 ,075f
 ,08d3
 ,00ff
 ,0105
 ,0105
 ,0615
 ,0877
```

6.3 Logging software

The logging program is written in C and was extended from the sfListen sample program. The program requires the following command line arguments:

host: Host name or IP address of computer with which to open a TCP/IP connection.

port: Port number of the above host with which to connect.

lowest ID: Lowest node ID number of sensor motes in the network. The system assumes consecutive ID numbers for sensor motes.

number of sensor motes: The number of sensor motes in the network is used to determine the number of buffers and files required.

The program connects to the base mote through the serial forwarder program using the host and port number received from the command line.

The program will loop, storing timestamp packets and the packet count and data from the data packets in the buffers. Each data packet has its data stored at a position in the relevant mote's buffer according to its packet number, so that data packets can be received in any order and easily output to file in order of packet number. This is particularly useful for adding data from missing packets.

The program will remain in this loop until a control packet is received to cause the program to get the current computer timestamp, create the files using this timestamp as the base of the file names, and write the contents of the buffers to the files. The data files will have -L appended to the file name if any missing packets were detected in the sample session. The contents of the data files are comma separated values as shown above, making it simple to import the data into a spreadsheet or other analysis program. The program will repeat until stopped by the user.

6.4 Sensing

The sensor motes are all Crossbow TelosB motes running a TinyOS application. The TelosB motes have 6 of the 8 ADC channels connected to header pins. The 2 channels not available at the header are channels 4 and 5, making the available channels: 0, 1, 2, 3, 6, 7.

Preliminary testing was performed using sensor boards developed by CPUT students. A TelosB plugin sensor board with 3 axis accelerometer and 3 axis magnetometer by J. Kleynhans was one such used [42]. It became evident that there were two issues with a plugin board. Firstly, the weight of the mote affected the accelerometer readings as it acted like a mechanical low pass filter for vibrations. The weight also made it difficult to attach the sensor mote to the transformer tank. Secondly, having a plugin board would require more motes, as each mote could only monitor a single point on the transformer.

It was decided to separate the sensors from the motes to address these issues. Each sensor was mounted on a small PCB that was attached to the transformer tank with thin double sided tape.

There are two sensor configurations. The single axis sampling uses 5 channels per mote, sampling only the z axis of each sensor. The 3 axis sampling uses all 6 channels available at the header, enabling 2 accelerometers per mote.

6.4.1 Sensor choice

The following attributes were considered when selecting accelerometers to use:

- Sensor frequency bandwidth,
- Operating voltage range,

Table 6.1: Selected attributes of accelerometers considered for use.

Attribute	ADXL335 [4]	LIS344ALH [77]	LIS352AX [78]
Sensor bandwidth	1.6 kHz, 550 Hz	1.8 kHz	2 kHz
Operating voltage	1.8 V–3.6 V	2.4 V–3.6 V	2.16 V–3.6 V
Acceleration range	±3 g	±2 g/±6 g	±2 g
Supply current	350 μA	680 μA	300 μA
Axes	3	3	3

- Acceleration range, and
- Supply current.

Initially it was uncertain how much bandwidth would be required for capture. It was therefore decided to select an accelerometer with a wide bandwidth—greater than 1 kHz.

The motes have an operating voltage of typically 3 V with a maximum of 3.6 V down to a minimum of 2.2 V, the minimum operating voltage of the ADC [82]. Without the ADC, the micro-controller will operate down to 1.8 V but the radio IC has a minimum operating voltage of 2.1 V [81].

The acceleration measured did not require a large range and the higher sensitivity of a lower range accelerometer was preferred.

It is always important to keep in mind the current consumption of all components in a sensor network as devices will likely be powered by batteries or small energy harvesting systems.

The LIS352AX from ST Microelectronics was chosen because of the attributes listed in Table 6.1 [78]. A number of accelerometers have z axis bandwidths far more limited than those of the x and y axes like the ADXL335 in Table 6.1. The x and y axes have a 1.6 kHz bandwidth, while the z axis has a 550 Hz bandwidth. A strong advantage of the LIS352AX (and similar ADCs from the same manufacturer) was the equal sensor bandwidth on each axis.

6.5 Summary

The system makes use of sensor motes running a simple application that takes its cues from the base mote. The sensor motes sample multiple channels in quick succession at a sampling frequency dictated by the base mote. The accelerometers, mounted

on small PCBs, are attached to the sensor motes by lightweight wires to prevent the weight of the mote from affecting readings, and to allow each mote to monitor multiple points. The base mote controls the network and enables reliable data collection. Data collected are logged to comma separated value files by a data logging program on a computer.

Chapter 7

Network protocol analysis

This chapter presents an evaluation of the the network protocol. Aspects of the protocol were simulated and results compared with measurements from empirical tests.

7.1 Network simulation

Portions of the protocol were simulated to evaluate their operation. Four network simulators were considered: ns-2 [5], OMNeT++ [68], Castalia [66] and Tossim [49]. ns-2 and OMNeT++ are both general purpose network simulators, while Castalia and Tossim are both sensornet specific. Castalia is built on OMNeT++ and is used for simulating sensornets, body area networks and other low power networks. Tossim is a simulator for TinyOS applications. Tossim is limited to simulating only individual TinyOS applications for the MicaZ platform. This meant combining the 2 programs that made up the network and removing all platform dependent code (though both MicaZ and TelosB have the same radio IC) before compiling for MicaZ and the simulator. ns-2 does not directly include sensornet radios, though there are contributed IEEE 802.15.4 radios. Many ns-2 sensornet protocol simulations in the literature use an IEEE 802.11 radio.

Castalia was the simulator chosen as it is a fairly comprehensive sensornet simulator. Castalia defines a number of components and parameters that form part of the complete network simulation. The root of the simulation is the sensornet component, where the number of nodes, their positions and similar parameters are set. The nodes comprise components including the application to be run and the communication components: the radio, MAC and routing layer. The nodes are connected through a

wireless channel. Castalia even supports the simulation of sensing.

The simulation implemented was designed to closely resemble the operation of the sensornet application on a TelosB platform running TinyOS. The main components involved are shown in Figure 7.1.

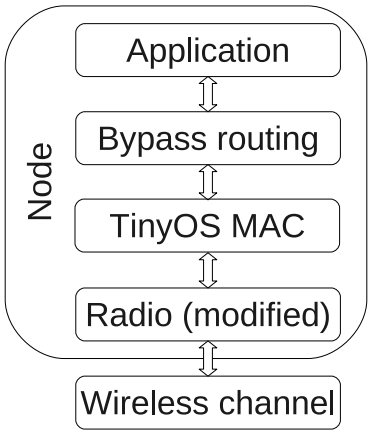


Figure 7.1: Main components for the Castalia sensornet simulation.

The wireless channel was modelled using a log-normal model [8] as shown in Equation 7.1. This is the default for Castalia. $PL(d)$ is the path loss over distance d. The reference distance (of 1 metre) is d_0, while n is the pass loss exponent. X_σ is the shadowing, a zero-mean random value of specified standard deviation (σ).

$$PL(d) = PL(d_0) + n10\log_{10}\left(\frac{d}{d_0}\right) + X_\sigma \tag{7.1}$$

TinyOS implements split-phase operation, where long running operations can be signalled to begin and will notify the calling component once the operation is complete. The Castalia radio component was modified such that the radio would signal the upper layers once transmission was complete, mimicking the TinyOS radio transmission split-phase operation.

Castalia does include an IEEE 802.15.4 MAC layer, however it only supports beacon enabled networks. A highly tunable MAC layer is provided. This was modified to create a TinyOS MAC. TinyOS introduces a short delay after a clear channel assessment (CCA) showing the channel to be clear, in case an acknowledgement may be on its way. The channel is assessed again after the delay prior to transmission. Many MAC layer parameters, along with parameters for the other components, were set in the simulation initialisation file.

The single hop network requires no routing, therefore the basic bypass routing component was used. It simply forwards messages between layers.

The application was based on Castalia's throughput test application and was configurable. The operation, for example, to determine data packet loss is as follows. The base mote sends a request-for-sampling packet. On reception, the sensor motes wait the approximate time for sampling to have completed. They then transmit the data packets as per the proposed protocol operation.

For all simulations, unless otherwise specified, there were 16 nodes (15 sensor motes and 1 base mote) in a 4 m $\times$ 4 m grid. The typical wireless channel used the defaults provided by Castalia as they are based on empirical tests [66]. An ideal channel was modelled as having a σ, bidirectional σ (variability of signal fading of the path in the reverse direction), and $PL(d_0)$ of 0 dBm. The radio and MAC layer's parameters were selected by studying the TinyOS source code and CC2420 radio datasheet. The simulation was repeated 40 times.

7.2 Empirical test configuration

Packet transfer was independently monitored for fault finding purposes during testing. Radio transmissions were logged to a file by using a separate mote running the standard TinyOS basestation application whose output was sent to the TinyOS listen utility. This version of listen had been modified to insert computer timestamps for each packet logged. This independent monitoring had proven useful in catching errors during development.

The output of the base mote was logged, using the same listen utility, to assist in fault finding the data logging element of the system. The network, including monitoring elements, is shown in Figure 7.2. Fifteen sensor motes were used in these tests.

Analysis of these monitoring elements' logs provided insight to the suitability of the network protocol chosen. The analysis in this chapter was performed on 38 consecutive cycles of operation, unless otherwise specified. One cycle of operation is considered to be 1 complete loop of the base mote's state machine. Each cycle contained 1 sample set from each active sensor mote.

The radio logs contain instances of all packets sent over the air, however the monitoring mote was susceptible to radio channel conditions and buffer overflows independent of the system base mote. As such, the number of packets logged was not identical to that of the system base mote.

The serial logs only contained packets sent from the sensor motes during the data collection and data recovery states and the completion signal packets. These were

the only packets sent from the system base mote to the system logging software, and therefore the monitoring logging program.

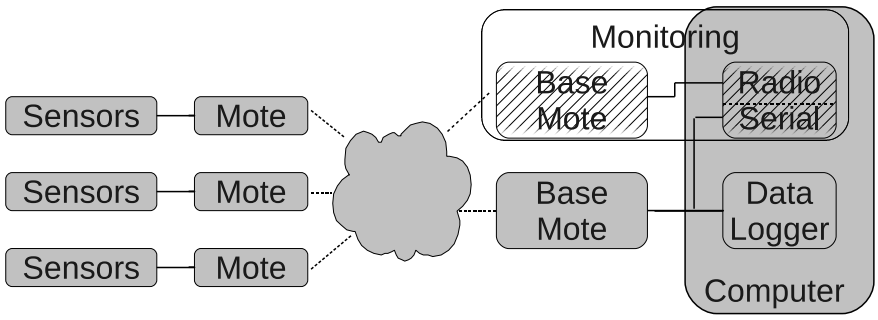


Figure 7.2: Components of the empirical test environment.

7.3 Packet transmission

The total packets sent during a cycle can be shown by the following equation:

$$P_{total} = P_{control} + P_{FTSP} + P_{timestamp} + P_{data} + P_{recovery}, \tag{7.2}$$

where:

- $P_{control}$ are the number of control packets sent. These were sent by the base mote to convey sampling information and to initialise the network. Sensor motes responded to the network initialisation control packet by sending another control packet. The base mote periodically sent network initialisation control packets until it had received responses from all motes.
- P_{FTSP} are the number of FTSP packets sent by the base mote. The mote was configured to send 10 packets per cycle.
- $P_{timestamp}$ are the amount of packets containing FTSP timestamps. Each sensor mote sent one just prior to sending the data packets.
- P_{data} are the data packets sent by the sensor motes. The total data packets (P_{data}) sent by a sensor mote during a cycle is $P_{dc} + P_{dr}$, where P_{dc} is the number of data packets sent during the data collection phase and P_{dr} is the number of resent data packets during the data recovery phase.
- $P_{recovery}$ are the number of packets sent by the base mote to initialise data recovery. Each packet was addressed to a single mote and contained up to 10 data packet requests.

Table 7.1: Packet types and the motes from which they originate.

	Base mote	Sensor mote
$P_{control}$	Yes	Yes
P_{FTSP}	Yes	No
$P_{timestamp}$	No	Yes
P_{data}	No	Yes
$P_{recovery}$	Yes	No

The packet types and which node generated them are summarised in Table 7.1.

The minimum total packet count for successful operation of the protocol with 15 sensor motes can then be calculated. The base mote would have sent 2 control packets, 1 for network initialisation and 1 to request sampling, while the sensor motes each would have responded to the network initialisation packet ($P_{control} = 2+15$). The base mote was programmed to send 10 FTSP packets ($P_{FTSP} = 10$) and the sensor motes each would have sent a single timestamp packet ($P_{timestamp} = 15$) before transmitting their 300 data packets ($P_{data} = 300 \times 15$). Therefore $P_{total} = 17+10+15+4500+0 = 4542$ packets. This gives a ratio for non-data packets to data packets of 42:4500 or 1:107.14.

The ideal sensornet from above can be expressed as having a goodput of 4500 packets/cycle using a broad definition of goodput as the number of data packets received (and logged) per 1 cycle. The goodput ratio, define here as the goodput divided by the throughput (total packets sent during 1 cycle) and expressed as a percentage, is 99,1 %.

The goodput remains the same for each cycle provided the losses do not exceed that which can be recovered. The throughput, and therefore the goodput ratio, are affected by packet loss. Data packet losses cause the base mote to send recovery packets. A single recovery packet is sent per sensor mote to request up to 10 data packets to be recovered. The network initialisation packets are the remaining packets whose count increase with detected loss.

The throughput for the scenario where a single lost data packet is detected per mote was calculated to be 4572 packets/cycle. In this scenario, the base mote had to generate 1 recovery packet per sensor mote, while each sensor mote had to resend 1 data packet. The goodput ratio is 98.43 %.

7.4 Packet loss

7.4.1 Control packets

Data from the radio logs show that, on average, 1.16 network initialisation control packets were sent by the base mote during a cycle. On average 1.12 control packets where detected by the monitoring mote. Sensor motes only send control packets in response to network initialisation control packets. The lower number of responses counted is most likely due to interference, causing the base mote to resend its requests, (though some losses may have occurred at the monitoring mote) as all sensor motes should have been active during the network initialisation phase.

7.4.2 Data packets

Each sensor mote always sends 300 data packets, though not all packets survive to be logged, therefore packets need to be recovered. For the cases where all data packets transmitted are successful received or too many data packets are lost, then $P_{data} = P_{dc} + 0$. In the current network configuration, this leads to $P_{data} = 300$ packets per sensor mote or 4500 packets for the complete network of 15 sensor motes. The sensor motes have consumed the same amount of energy for both cases, though the usable output from each case differs considerably. Typically, though, there is some data packet loss per cycle.

The data collection phase was simulated to look at data packet reception (throughput and loss). The operation was simulated with an ideal channel and the typical channel. The effect of roughly synchronised transmission versus passive congestion control through staggered transmissions was considered. Figure 7.3 shows the mean results of the simulation. The mean losses with no congestion control were 20 packets (0.444 % loss) and 31.875 packets (0.708 % loss) for the ideal channel and typical channel respectively. An increase in path loss attenuation of 5 dBm with no congestion control resulted in an increase to 141.65 packets lost (3.148 % loss) as seen in Figure 7.4.

Congestion control was managed passively by having each sensor mote start transmitting during the data collection transmission state at an offset defined by the number of sensor motes multiplied by a delay value. A delay value of 10 ms was chosen empirically as it showed reduced losses while maintaining rapid transmission. A comparison of the effects of a change in the delay parameter under typical channel conditions is shown in Figure 7.5.

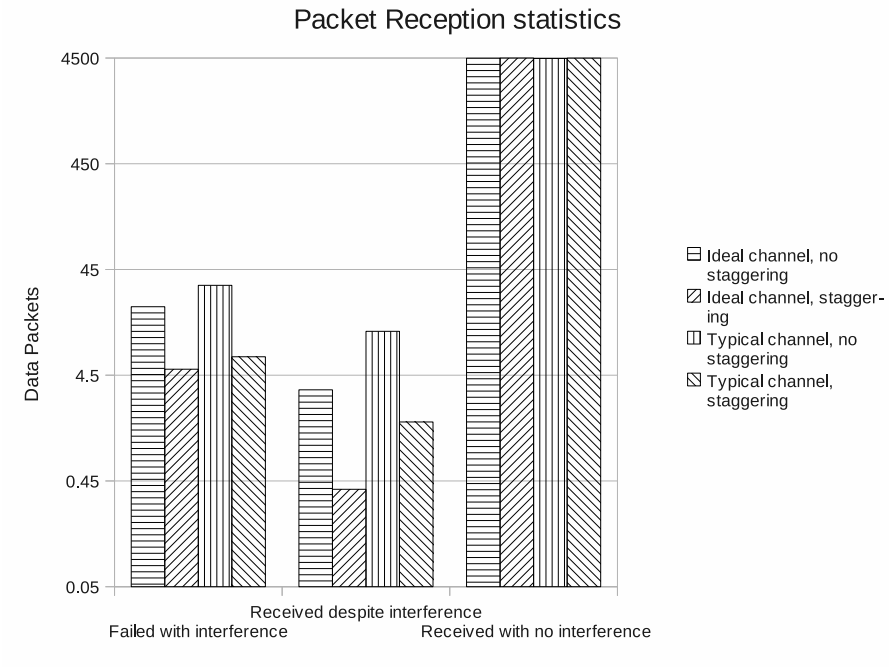


Figure 7.3: Data packet reception statistics for the network with and without passive congestion control (transmission staggering) on an ideal channel and typical channel.

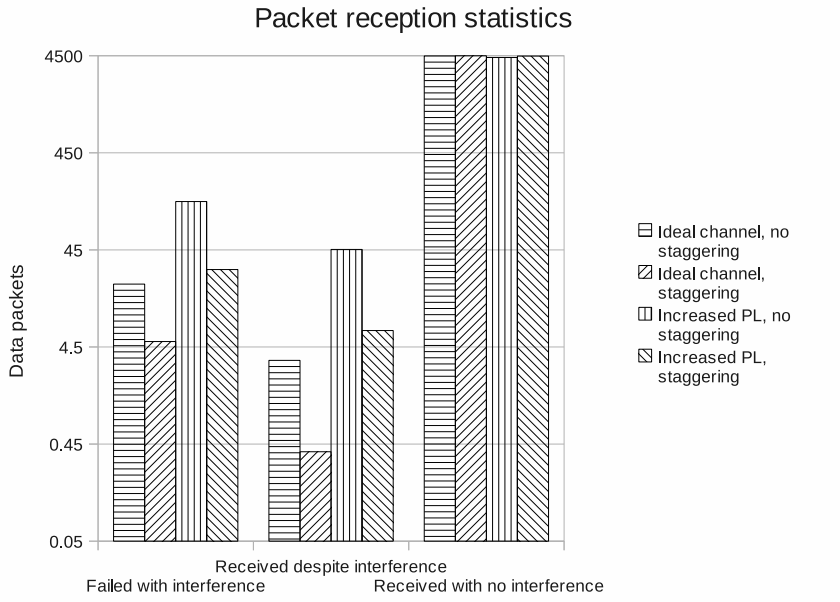


Figure 7.4: Data packet reception statistics as per Figure 7.3, except that $PL(d_0)$ has been increased 5 dBm from that of the typical channel.

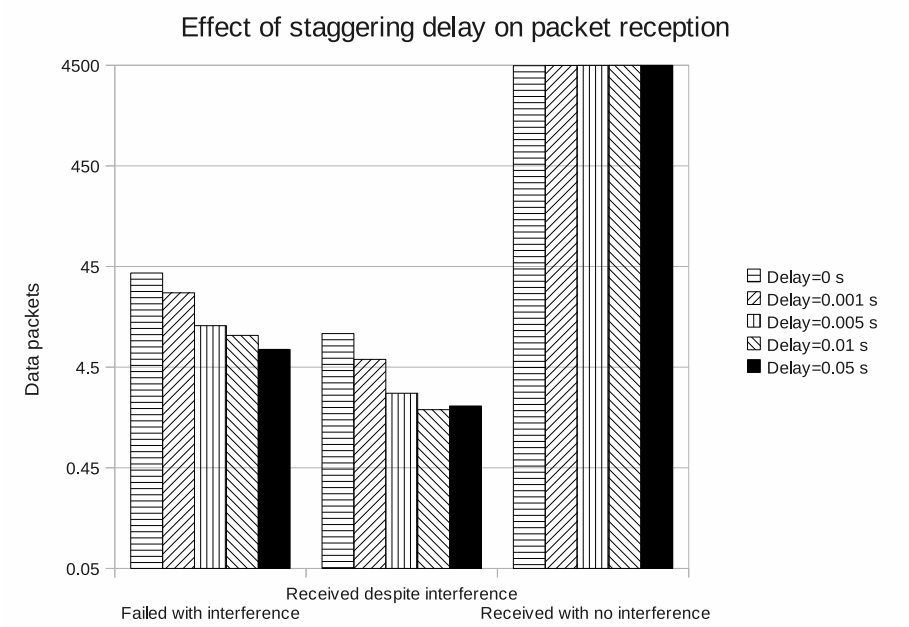


Figure 7.5: The effect on data packet reception of the initial transmission offset with various delay constants.

The serial logs were analysed to determine the number of data packets received during normal operation. The mean number of data packets received during normal operation over 38 sample sets was 4366.03, with a standard deviation of 53.97. Therefore the mean packet loss during data transmission was 133.97 data packets or 2.98 % of all data packets sent. The largest loss during this period was 397 data packets or 8.82 % of all data packets sent. This large loss was attributed to a temporarily faulty mote that did not send any data packets during a single cycle. The largest total loss ignoring the cycle with the faulty mote was 202 data packets. The mean packet loss per sensor mote was 7.98 data packets (σ 2.36 data packets) excluding the single 100 % loss. The single largest data packet loss experienced by an individual mote, excluding the single 100 % loss above, was 32 data packets or 10.67 % data packet loss, 2 packets over the threshold for data packet recovery. Only this sample set and the faulty mote set experienced losses beyond the recovery threshold for the 38 sample sets.

The base mote sends, on average, 1.36 data recovery packets per sensor mote per cycle. The mean total recovered packets per cycle was 126.28, less than 10 packets per node. Therefore packet loss is not distributed evenly, but some motes experience heavier loss than others. The mean recovered packets is less than the mean loss stated above as it did not include the packets from the 2 cycles with heavy losses.

7.5 Causes of packet loss

Packet loss can be attributed to interference and losses on the radio link and flow control problems through buffer overflows on the base mote. By comparing the analysis of the serial logs to similar analysis of the radio logs, it was possible to determine the cause of data packet loss. As the radio monitoring used an independent mote logging network traffic, if the radio logs showed more data packets received before recovery began than the serial logs, then the difference in packet counts represents losses at the base mote rather than channel interference or collisions over the air. Radio logs cannot be used to gain exact figures as the independent mote is subject to similar flow control problems as the base mote. No useful information can be gained from cases where the radio logs show higher losses than the serial logs as the reason for these higher losses cannot be determined without additional data. The radio logs can, however, give a fair reflection of the losses occurring at the base mote versus losses occurring over the air.

Using the same 38 sample sets, taking the difference between the data packet counts from the serial logs and the radio logs and only using the values where the packet loss was lower for the radio logs, the following was observed:

- 22 of the 38 cycles had flow control issues.
- The total mean loss for these 22 cycles was 125.86 data packets (σ 63.05 data packets).
- The total mean loss for these 22 cycles due to flow control issues was 14.05 data packets (σ 13.05 data packets), or approximately 11 % of lost data packets.
- The minimum and maximum lost data packets due to flow control issues in these 22 sample sets were 1 and 51 respectively.

7.6 Latency

Data packet latencies were considered from the perspective of the application layer. It can be seen from the simulation results in Figure 7.6 that the vast majority of packets were delivered promptly, within approximately 7 ms, as to be expected for a single hop network. All data packets were delivered within 13.333 ms when transmission staggering was implemented. The small spread can be attributed to the operation of

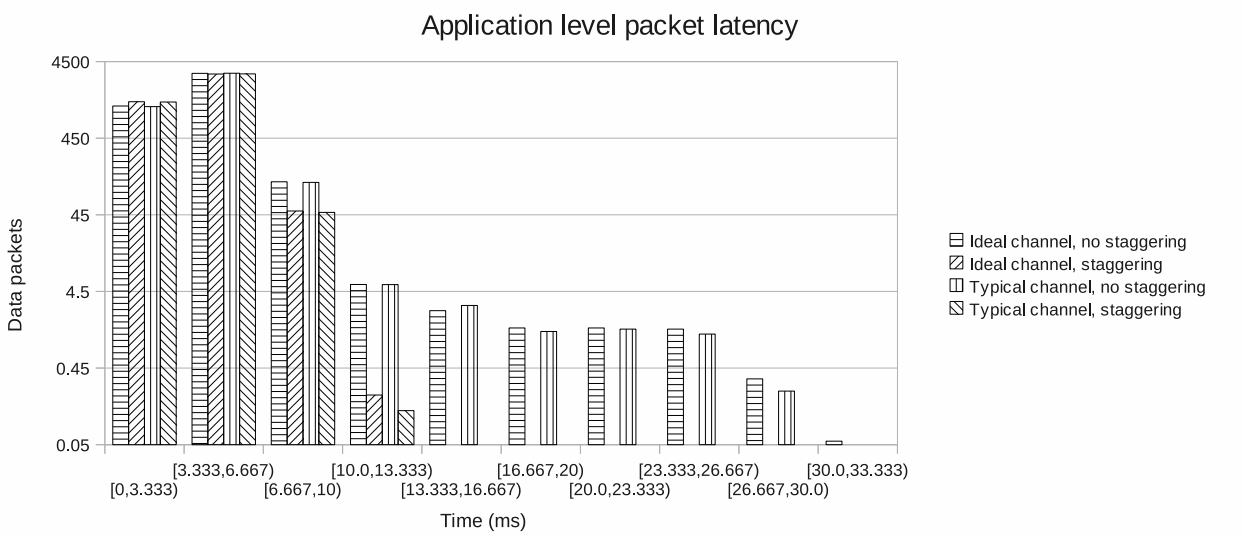


Figure 7.6: Application layer latencies for data packets.

the CSMA algorithm. The effect of the CSMA back-off algorithm can be seen most prominently in the plots with no transmission staggering.

The serial and radio monitoring logs were used to estimate latencies in the empirical tests. The difference in time between the logging computer generated timestamps for received packets with consecutive sequence numbers was calculated. Only consecutive sequence numbers were included to avoid large time differences due to missing data packets. These time differences represent delays introduced in the application, TinyOS processing delays and CSMA and propagation delays. In addition to these delays, there are base mote processing delays (including data packet buffering), serial data transfer and delays in presenting the data from the serial port to a TCP/IP socket before it can be timestamped by the logging program.

The time differences of the serial and radio logs were closely correlated. Data from the serial logs were chosen as these timestamps represented that of packets that had been passed through the base mote under observation rather than an independent monitoring mote.

An estimate of the latency was derived by subtracting the inter-packet transmission delay of 15×11.7ms (175.5 ms) from the time differences. This yielded a mean of 11.95 ms, σ 31.96 ms. These values contain a number of delays internal and external to the observed system as stated above, though they do provide a glimpse in to the transmission latencies.

7.7 Conclusion and summary

Simulation and empirically gained results of the network protocol were presented. The protocol had a high data packet to control packet ratio. The data packet losses experienced in the empirical tests were higher than those of the simulation under similar conditions, though well within an order of magnitude. A higher loss rate for empirical tests is to be expected due to the limitations of the simulation and numerous factors at play in the lab experiments. The simulation typically has immediate transitions between events, which is not the case in reality. Channel conditions and the presence of Wi-Fi signals may have contributed to the increase in loss. The channel at which the primary Wi-Fi source operated was, however, offset from the channel that the motes were using.

Chapter 8

System verification

The system was tested to verify its operation according to the design and to quantify the accuracy and consistency of the results obtained. A network was created using 15 sensor motes. These sensor motes could be configured, through a command from the base mote, to use up to 6 analogue input channels. A maximum of 6 channels was a hardware limitation of the TelosB motes. Six channels allowed for 2 3-axis accelerometers to be attached to a mote.

Two sets of tests were performed. The first tested the bare system with no sensors attached. The second tested the system with sensors attached in a controlled environment. Additionally, the system with accelerometers was applied to a 3 phase, 38 kVA / 47 kVA, 380 V / 220 V, 50 Hz, AC isolation transformer. This chapter describes the tests performed on the system without sensors, and presents the results obtained.

8.1 Test configuration

The first tests were run using either an Agilent 33220A 20 MHz function generator or a voltage divided output from a battery attached to the motes' ADC inputs. A Tektronix TDS 2024B digital oscilloscope was used to confirm signals. In all tests the motes were powered by batteries, unless otherwise stated. The data was analysed using MATLAB scripts. Original scripts were written by Dr Jevon Davies and adapted by the author.

Tests were run using the internal 1.5 V ADC reference as experiments on a transformer, shown in the following chapter, showed the output voltages from the accelerometers to be very small, but with a DC offset. As such, the 1.5 V reference

voltage provided better resolution for the small input signals than the internal 2.5 V ADC reference. The 1.5 V internal reference was stable at a supply voltage down to 2.2 V, whereas the 2.5 V internal reference voltage only allowed for a supply voltage down to 2.8 V [82]. The 1.5 V reference was therefore the better choice for both small input signals and long term battery operation.

Each sample set contained 3000 samples divided by the number of channels used, e.g. 600 samples per channel for 6 channel operation. Six channel operation was used for all tests. In all sample domain plots, only a small, contiguous set of samples were chosen to illustrate the waveform. These samples were all chosen from midway through the sample set in time. The mean had been subtracted from the amplitude values, unless otherwise stated. Frequency spectra were generated by applying a Fast Fourier Transform (FFT) over all but the first 100 samples. The large number of samples were required to generate accurate frequency spectra without further increasing the sample rate.

8.2 Sample period

The sensor mote software used a 32 kHz clock through the Alarm32khz32C component and Alarm (and Init) interface for the sample timer. The 32 kHz clock had an external 32.768 kHz crystal as its source. A timer value of 32768 represented 1 second, therefore the timer incremented in 30.5 μs steps [73]. To achieve a sampling frequency of at least 1 kHz would therefore require a timer value of 32 giving a sample period of 976.6 μs—a sample frequency of 1024 Hz. However, it was shown empirically that a timer value of 32 provided a sample frequency of less than 1 kHz. To determine the actual sample period, a pin was toggled once per sample period. The generated train of pulses was captured by the oscilloscope and saved to a USB memory stick. The individual sample periods could be determined by measuring the time between transitions. The mean sample period measured over 10 half-periods for 32 clock cycles was 1006 μs. The mean sample period for a timer value of 31 was 976 μs, while it was calculated to be 946 μs. Both sample periods were approximately 30 μs longer than their calculated values. It is assumed that the difference was the overhead involved in using the timer. A captured waveform for 31 clock cycles is shown in Figure 8.1.

It can be seen from the figure that after the fourth transition there is a low level whose duration is twice that of the others. This occurred when the first 20 or more samples were removed from the buffer. This longer low level was not present if the dequeue loop was prevented from running. Checking the generated C code and the nesC

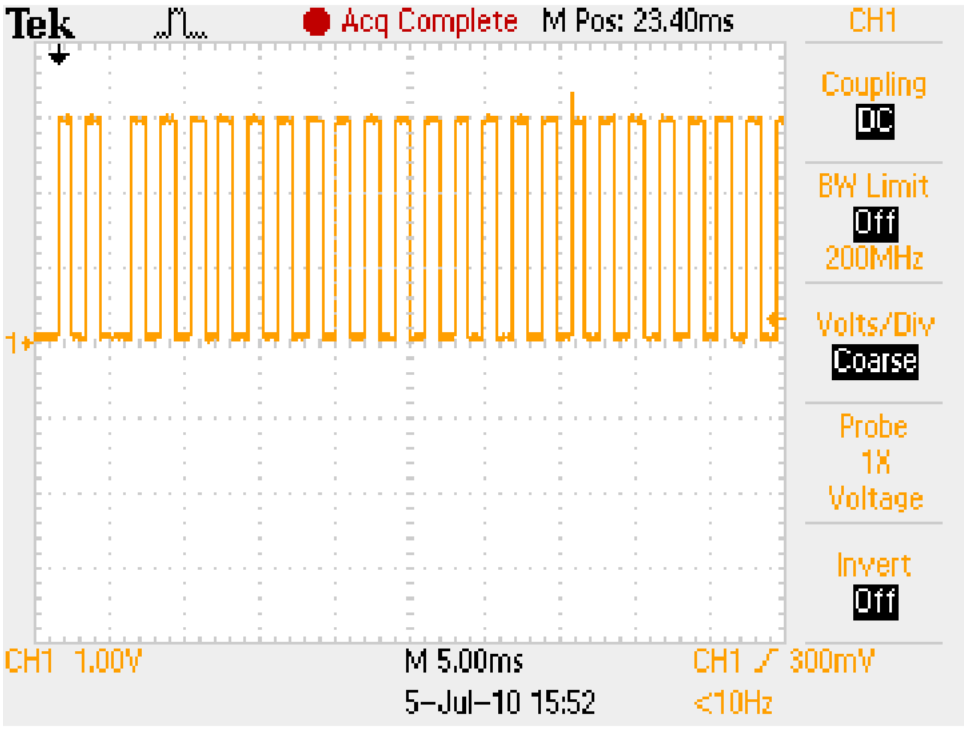


Figure 8.1: Pulse train from toggling a pin each sample period with a timer value of 31.

source, showed that the function in which the dequeue loop was run, was considered atomic. All instances of the atomic keyword within this function were replaced with the following comment: `/* atomic removed: atomic calls only */` in the generated code, as they were redundant. The function would therefore run to completion with interrupts disabled.

8.3 Effect of supply voltage on sample values

The sensornet nodes were expected to operate for long periods of time and be powered from batteries or renewable energy sources. Both energy sources would cause the nodes to experience a change in supply voltage over time. The frequency and amplitude of a signal were compared at different supply voltages to see the effect of the voltage change.

In the experiment a 100 Hz, 500 millivolts peak-to-peak (mVpp) sine wave from the function generator was connected to the ADC channels on a mote. The mote was powered by a DC power supply with sets of measurements taken at different supply voltages. Initially, to get a baseline, the mote was powered by USB. Two other motes

Table 8.1: Peak-to-peak recorded voltages at different supply voltages.

Source	USB	2.9 V	2.8 V	2.5 V	2.2 V
ADCpp (ADC units)	2728.20	2725.80	2725.30	2731.10	2727.40
σ (ADC units)	14.97	6.68	5.21	10.48	4.79
Vpp (V)	0.999	0.998	0.998	1.000	0.999

powered by USB were used to complete the network and logged data was analysed using MATLAB. The mean of the peak-to-peak voltages across 10 sample sets was collected for each supply voltage level in addition to comparing the frequency of the signals.

Table 8.1 shows the results obtained for the peak-to-peak voltages (Vpp). The first row contains the mean ADC peak-to-peak (ADCpp) values for each supply voltage. The amplitudes for the supply voltages in the acceptable supply voltage range (2.2 V minimum) were within approximately 0.2 % of each other. The ADC values were converted to volts assuming a stable 1.5 V reference at each supply voltage.

If the baseline measurement is assumed to have a 1.5 V reference and the peak-to-peak input voltage is the same for each measurement set (0.999 V), then it can be determined that the reference voltage for the reduced supply voltages remains stable, only varying between 1.498 V and 1.501 V. This is within the expected range of the internal reference voltage. The frequency for each test was shown to be 99.9 Hz. This frequency was obtained in MATLAB, by applying an FFT to the sample domain data assuming a sample period of 976 μs. Four hundred sample points were used. The resolution of the frequency was then $1/(400 \times 976\mu s)$ or 2.561 Hz. By using a fairly long sample set, the sampled signal could be expressed in the frequency domain reasonably accurately.

8.4 Amplitude

A DC voltage source was applied to an ADC channel and the sampled values compared to the input signal to determine whether the sampled amplitudes of signals were accurate representations of the input signal amplitudes. The DC signal was supplied by a battery through a voltage divider with the output put through an operational amplifier (op-amp) buffer. The ADC reference voltage was 2.5 V and at 12 bit resolution; the ADC values were converted to volts by multiplying the value by 610.5 μV (2.5 V/4095). Table 8.2 lists the voltages measured by the oscilloscope and the voltages obtained from the mean of the sampled values assuming a reference voltage of 2.5 V.

Table 8.2: Comparison of input voltages to measured amplitudes with an ADC reference of 2.5 V.

Ideal input (mV)	10	50	100	500	1000
Measured input (mV)	10.2	50.1	100	501	1000
Mean (mV)	7.512	47.723	97.948	499.464	993.071
σ (μV)	65	53	88	189	112
% error	26.353	4.745	2.052	0.307	0.693

Table 8.3: Comparison of input voltages to measured amplitudes with an ADC reference of 1.5 V.

Ideal input (mV)	10	50	100	500	1000
Measured input (mV)	9.92	50.28	100	500	1000
Mean (mV)	6.859	47.034	96.466	494.879	991.535
σ (μV)	21	24	16	34	32
% error	30.857	6.456	3.534	1.024	0.847

The percentage error between the converted voltages and the voltages measured by the oscilloscope is shown. It can be seen that the measured amplitudes are less accurate at the lower input signal amplitudes with the 2.5 V internal reference.

The experiment was repeated using the 1.5 V internal reference. The results are shown in Table 8.3. Both tables show similar results, with highest inaccuracies at very low amplitudes, that can most likely be attributed to noise.

8.5 Frequency

Sine waves generated by the function generator were used to test the accuracy of the system with respect to frequency. A portion of the captured signal in the sample domain, and the Fourier transform of the signal for the 100 Hz and 300 Hz signals are shown in Figures 8.2 and 8.3 respectively. The mean has been subtracted from ADC values. The frequency values of the peaks for each Fourier plot are 99.9 Hz and 299.7 Hz, respectively.

The captured signal in the sample domain does not clearly resemble that of the input signal for Figure 8.3. As the frequency of the input signal tends towards the sampling frequency, there are too few samples to simply plot the waveform. However, there is sufficient information to recreate it as can be seen in the Fourier plot.

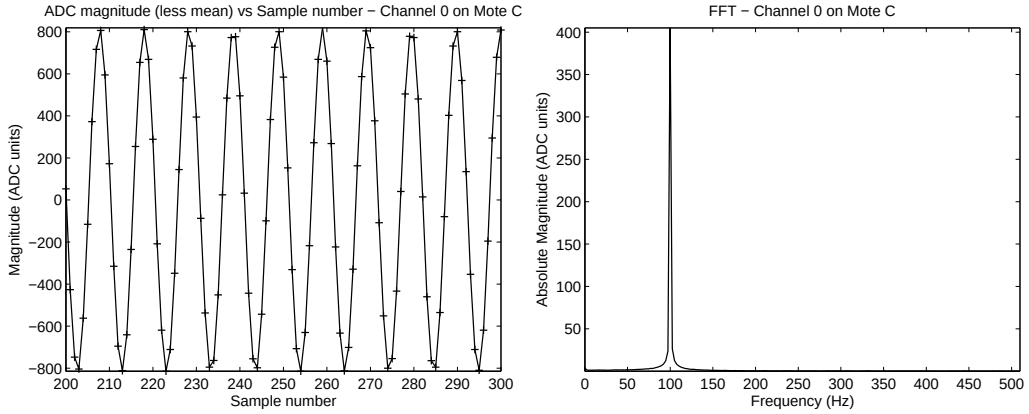


Figure 8.2: The sample domain plot and Fourier transform for channel 0 on mote C with 100Hz sine wave input.

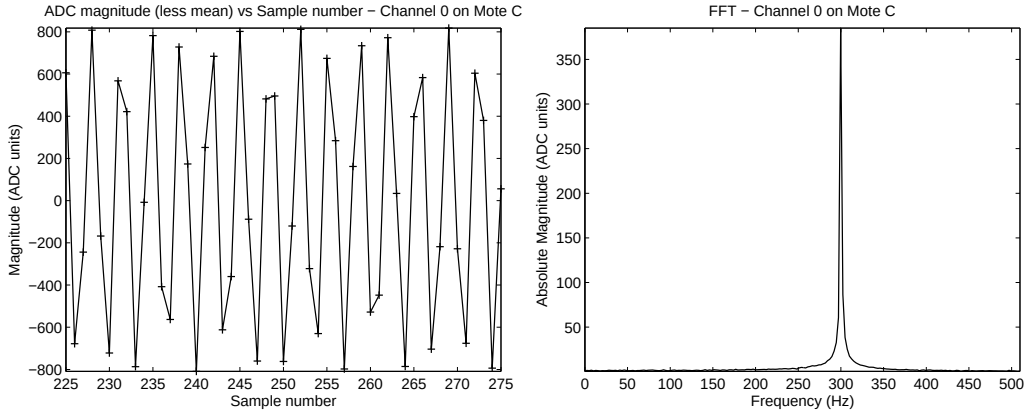


Figure 8.3: The sample domain plot and Fourier transform for channel 0 on mote C with 300Hz sine wave input.

8.6 Synchronisation

The manner in which the ADC was used for sampling means that there was a difference in time between the sampling of each channel at each sample point. The hardware did not permit simultaneous sampling of multiple channels.

The ADC sampling was done in *sequence-of-channels* mode using the internal ADC oscillator with the *multiple sample and convert* (MSC) bit set. This meant that the time taken between sampling each channel was the sum of the sample time (t_{sample}), the conversion time and the result storage time as can be seen in Figure 8.4. The conversion time and result storage time were set to 12 clock cycles and 1 clock cycle respectively, while t_{sample} was set by the user [80].

In addition to there being a difference in time between channels on a single mote,

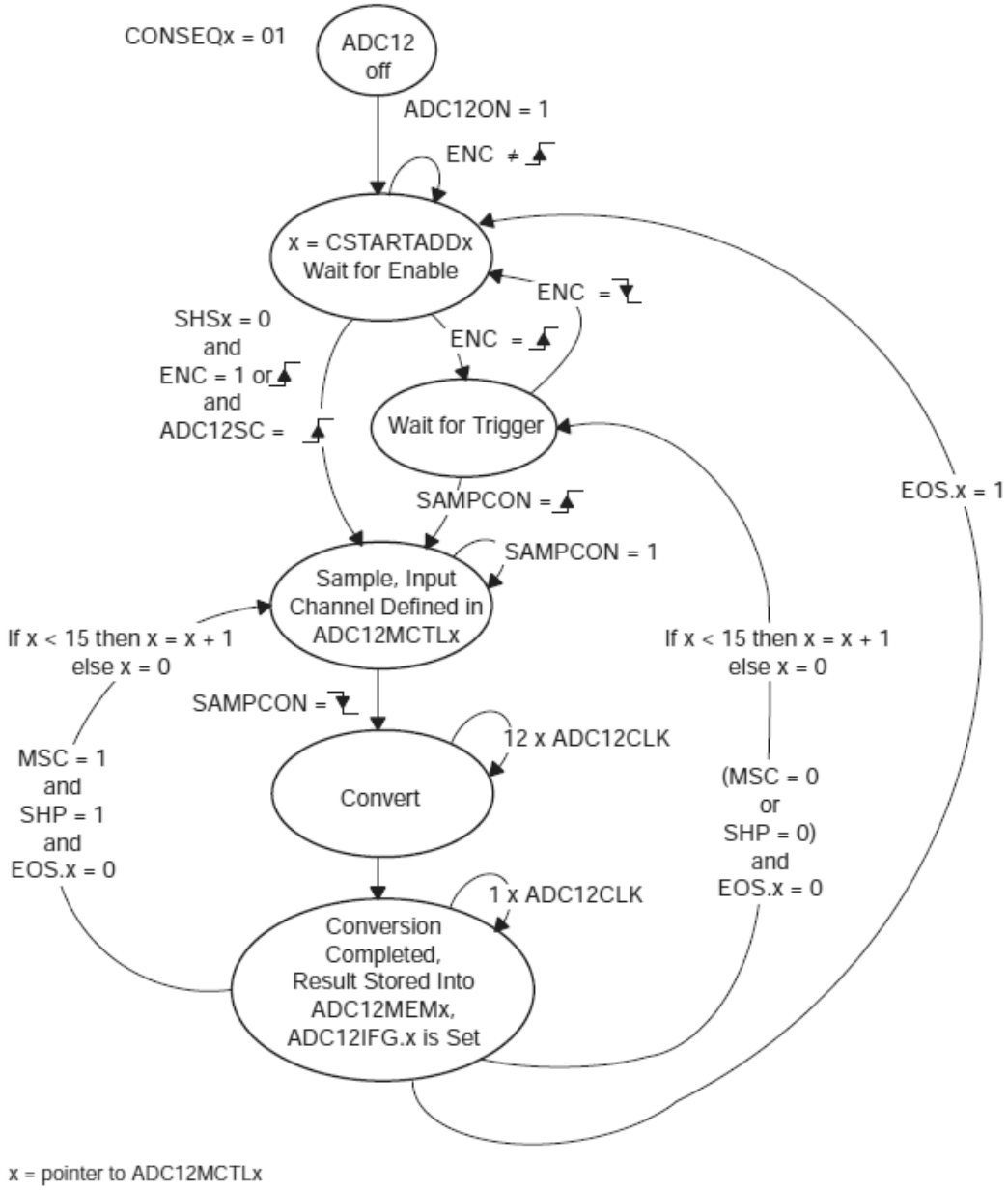


Figure 8.4: State diagram for *sequence-of-channels* mode. Reprinted from [80].

there was also a difference in sampling start times between sensor motes. Though the motes used the same hardware and ran the same software, minor differences in hardware components resulted in clock skew between motes. The motes were not started simultaneously, which resulted in offsets between timers.

Therefore, it was necessary to determine the the time difference between channels on a single mote (inter-channel) and between channels on different motes (inter-mote) for each sample point.

The sampled data provided an amplitude measurement, in ADC units, for each channel, for each sample point. Each sample point was 1 sample period apart in time.

Though the difference in time between channels could not be directly measured, ADC channels sampling the same signal have different amplitude values due to the differences in sampling times. If a linear signal is applied simultaneously to the ADC channels of each sensor mote, then the slope of the signal can be calculated. With the slope known, the portion of the sample period that is equivalent to a change of 1 ADC unit can be used to determine the timing between channels.

A 100 Hz triangle wave was applied simultaneously to the ADC channels of each mote. Consecutive sample points from parallel slopes were extracted from the sample sets. The difference between these sample points was calculated for channel 0 on each mote and can be seen in Table 8.4. The table shows the relationship between amplitude and time. These values were calculated using a sample period of 976 μs, as determined in Section 8.2.

The method used to obtain the values in Table 8.4, and the timing data from the table were used for calculating the inter-channel and inter-mote synchronisation.

8.6.1 Inter-channel synchronisation

The expected time between channels was t_{sample} plus the conversion time and result storage time. The MSP430 user guide lists the following equation for selecting an appropriate sample time (t_{sample}) [80]:

$$t_{sample} > (R_S + R_I) \times \ln(2^{13}) \times C_I + 800\text{ns}, \tag{8.1}$$

where R_S is the output resistance of the accelerometer, R_I is the input resistance of the multiplexed ADC, and C_I is the ADC input capacitance. R_I is given as 2 kΩ and C_I as 40 pF. The LIS352AX accelerometer has an output impedance of 32 kΩ [78] and so, from the equation, needs a t_{sample} of greater than 13.06 μs. A time of 13.06 μs is equal to 65.3 clock cycles of the 5 MHz ADC oscillator. The closest selectable number of clock cycles for a t_{sample} greater than 65.3 is 96 clock cycles, which was consequentially chosen.

The time between samples was therefore 109 ADC clock cycles (96 + 12 + 1). The ADC oscillator has an expected typical frequency of 5 MHz making the expected time between samples approximately 21.8 μs [80]. This meant that for a 500 Hz waveform, the difference in time between any 2 channels on the same mote would be 1.09 % of the period and the difference between the first and last (sixth) channel sampled would be 109 μs or 5.45 % of the period.

Table 8.4: Difference in amplitude between consecutive data points on a slope in ADC units and the relationship of amplitude to time, and its inverse, assuming a sample period of 976 μs.

	Mote A	Mote B	Mote C	Mote D	Mote E	Mote F	Mote G	Mote H
Mean (ADC units)	640.194	642.098	638.803	636.118	632.166	643.767	641.761	645.322
σ (ADC units)	11.173	10.809	9.752	8.504	10.994	10.418	8.185	8.134
Minimum value (ADC units)	475.000	486.000	465.000	501.000	462.000	454.000	495.000	501.000
Maximum value (ADC units)	714.000	711.000	688.000	673.000	746.000	724.000	683.000	674.000
Amplitude/time (ADC units/μs)	0.656	0.658	0.655	0.652	0.648	0.660	0.658	0.661
Time/amplitude (μs/ADC unit)	1.525	1.520	1.528	1.534	1.544	1.516	1.521	1.512
	Mote I	Mote J	Mote K	Mote L	Mote M	Mote N	Mote O	
Mean (ADC units)	652.474	640.647	635.102	633.027	633.649	639.320	638.922	
σ (ADC units)	9.668	10.226	8.474	9.093	12.350	11.104	9.935	
Minimum value (ADC units)	507.000	467.000	465.000	504.000	444.000	451.000	432.000	
Maximum value (ADC units)	706.000	730.000	676.000	706.000	776.000	757.000	713.000	
Amplitude/time (ADC units/μs)	0.669	0.656	0.651	0.649	0.649	0.655	0.655	
Time/amplitude (μs/ADC unit)	1.496	1.523	1.537	1.542	1.540	1.527	1.528	

The mean time between consecutive channels was calculated as 22.765 μs with a standard deviation of 10.932 μs. These calculations were made by taking the change in ADC values on parallel slopes between channels on mote A and applying the time per amplitude value for mote A from Table 8.4. This test was run on the 6-channel sensor mote software and used 20 sample sets. The channels used were those available at the extension headers. These were channels 0, 1, 2, 3, 6 and 7, and were sampled in that sequence. For simplicity, the channels will be referred to as 0–5 for the remainder of the document.

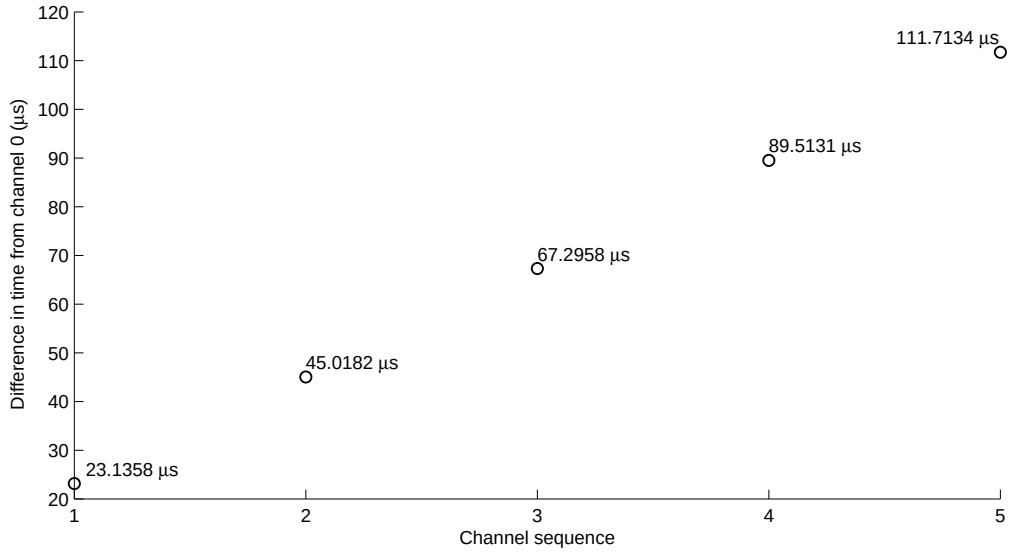


Figure 8.5: The difference in time between channels relative to channel 0.

Figure 8.5 shows the mean of the differences in time between the channels relative to channel 0.

There were 109 ADC clock cycles between consecutive samples. A mean time between channels of 22.765 μs gave an ADC clock time of 4.788 MHz. This was slower than the typical 5 MHz value but well within the specified range of values for the internal ADC clock.

Knowing the mean difference in time between channels enabled plotting the measured signal in the time domain, as opposed to the sample domain shown thus far. Figure 8.6 shows a plot of channel 0 and channel 5, the first and last channels sampled, in the sample domain and Figure 8.7 shows the same plot adjusted to the time domain. Comparisons of the signals can now be done relative to time.

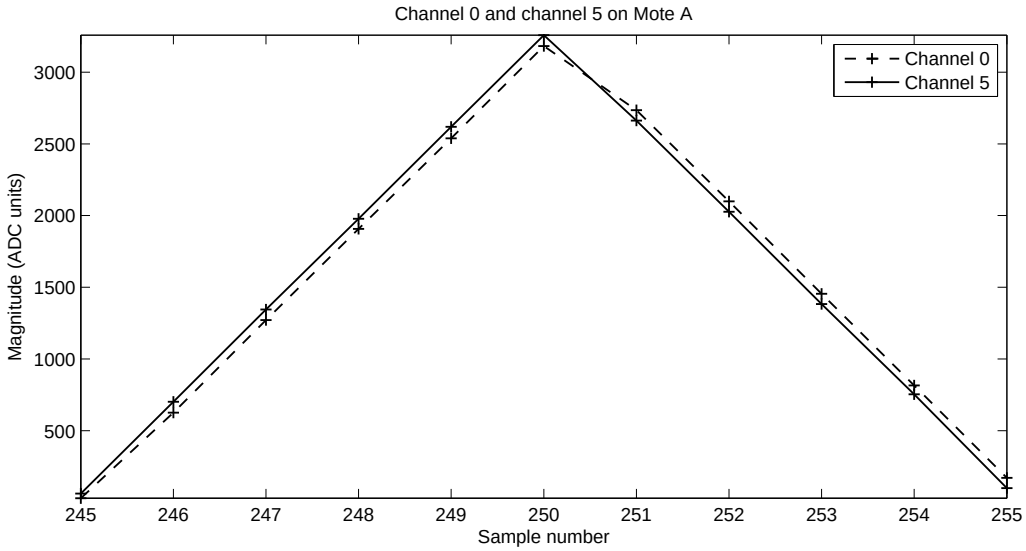


Figure 8.6: A portion of the measured signal as captured by channel 0 and channel 5 on mote A in the sample domain.

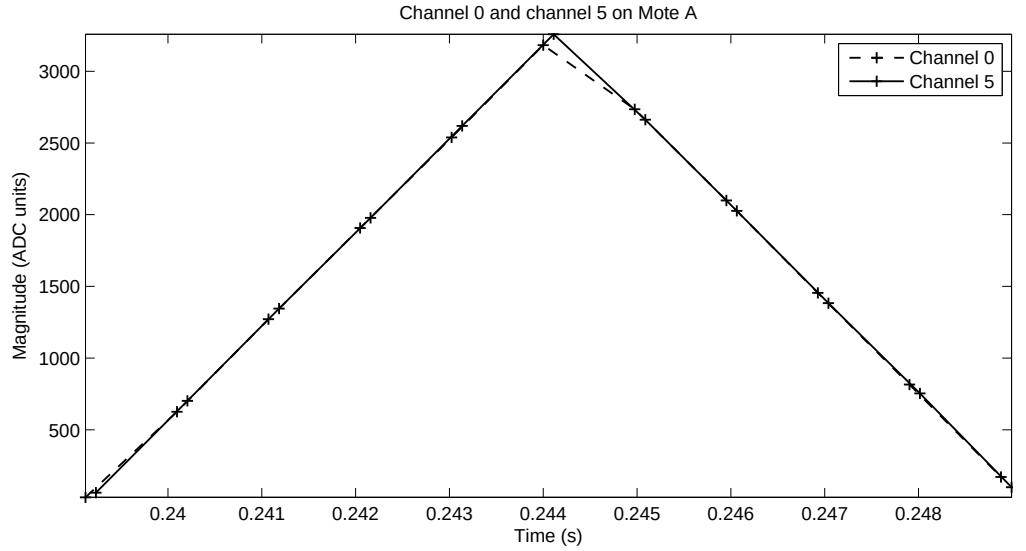


Figure 8.7: The same portion of the measured signal from Figure 8.6 transferred to the time domain.

8.6.2 Inter-mote synchronisation

Each sample set from each mote had an accompanying FTSP timestamp that represented the global time—generated by the sensor mote—before the start of a sample set. These timestamps were used to determine the sampling time offsets of motes relative to each other. However, these timestamps had a granularity of 30.5 μs.

The FTSP timestamp differences relative to Mote A's timestamp are shown in Figure 8.8. Eight consecutive sets of timestamps are shown, with 2 sets of timestamps per plot. The motes were typically within 1 to 2 clock cycles apart.

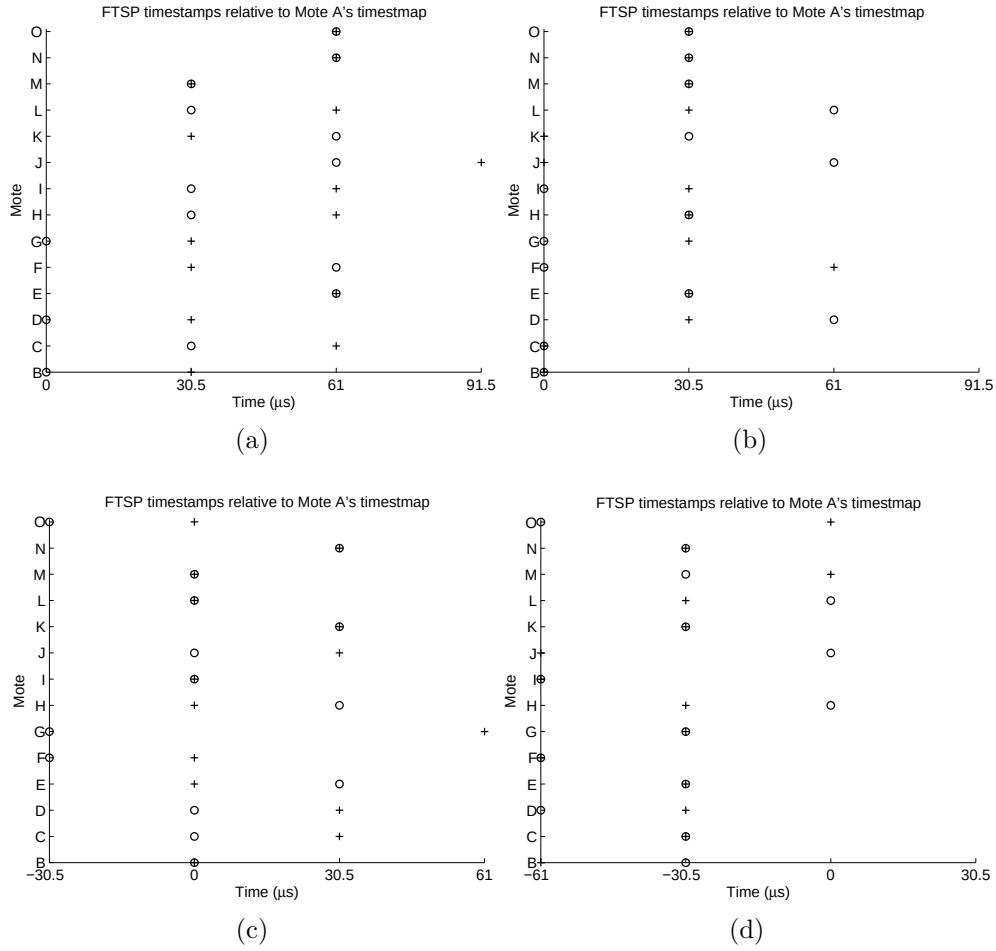


Figure 8.8: The FTSP timestamps relative to Mote A's timestamp are shown for 8 consecutive timestamps. Each plot shows 2 consecutive sets of timestamps, represented by the plus sign and circle, respectively.

An experiment was run to illustrate the use of time synchronisation. The test set up was as before, however, 2 motes were set to start 21 clock cycles early, while another 2 motes were set to start 21 clock cycles late. Twenty one clock cycles was chosen as it was approximately $1/16^{\text{th}}$ of the period of a 100 Hz signal. Figure 8.9 shows a portion of the sampled signal from channel 0 on each mote, arbitrarily chosen from

sample set 8. The signal is plotted in the sample domain. Figure 8.10 shows the same portion of the sampled signal plotted in the time domain. The accuracy is limited by the 30.5 μs clock unit.

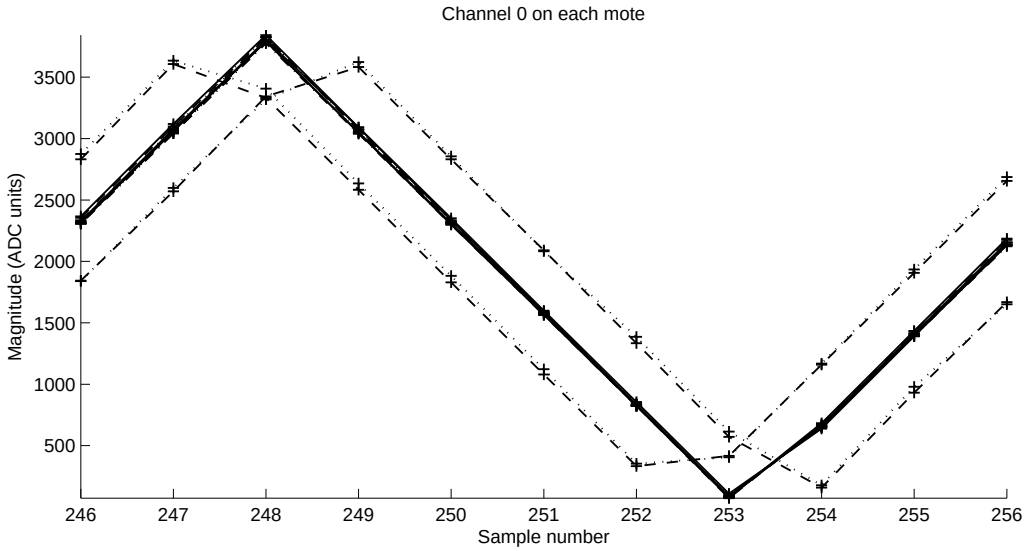


Figure 8.9: Sample domain plot of a portion of sample set number 8.

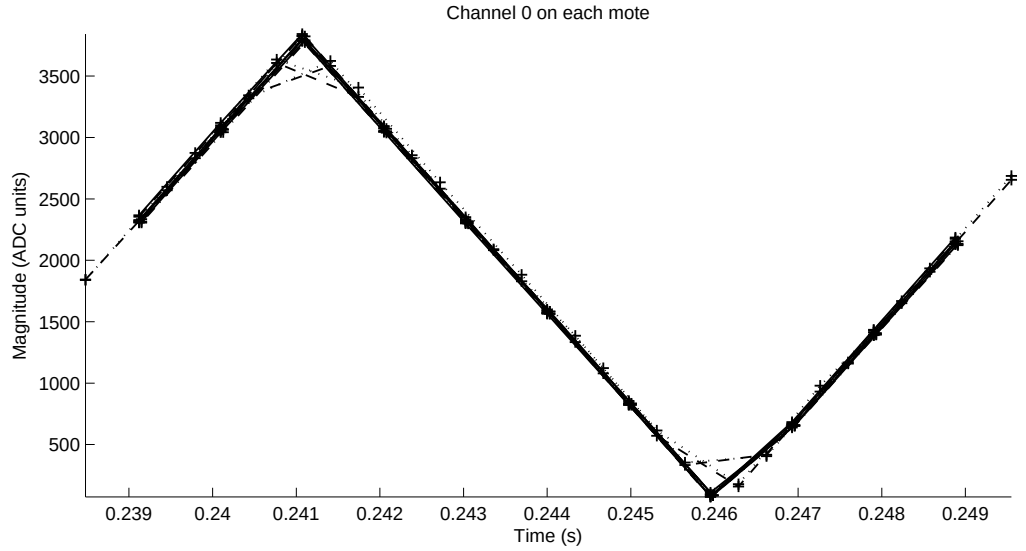


Figure 8.10: Time domain plot of the samples from Figure 8.9.

The difference in time between any 2 channels is then the inter-channel delay between the channels and the inter-mote offset taken from the FTSP timestamps.

8.7 Summary

The sensornet data logging system has been evaluated in terms of its ability to recreate the amplitude and frequency of input signals. The frequency is accurately captured but amplitudes are inaccurate at small amplitudes, as the signal-to-noise ratio decreases. Change in supply voltage has an insignificant effect on the accuracy, provided the supply voltage remains within the normal operating range. Synchronisation between channels on a mote and across motes has been quantified.

Chapter 9

Vibration measurement testing

These tests show the operation of the system with accelerometers attached. The system was tested in a laboratory setting on a vibrating platform, described below. The application of the system to transformer vibration measurement is also included.

9.1 Testing the system with sensors

These tests used an Agilent 33220A 20 MHz function generator as an input to a Crest Audio CA 6 amplifier. This setup was used to drive a vibration platform. The platform comprised a flat plastic sheet mounted on a box attached onto the cone of a speaker to provide a level platform for placing accelerometers and piezoelectric cantilevers for vibration testing. This vibration platform was developed by D. J. de Villiers [14] and can be seen in Figure 9.1. A Tektronix TDS 2024B digital oscilloscope was used for measuring the outputs from the accelerometers.

For the following tests, the sensors were placed on the vibration platform as shown in Figure 9.2. There were 6 3-axis sensors, 2 per mote, and the 6-channel sensor mote software was used. The synchronisation tests used 15 sensors, with 1 sensor per mote. The three central sensors had all 3 axes sampled, while the remaining sensors only supplied a z axis output for the synchronisation tests. The motes in the 15 mote tests were powered with AA batteries, while the smaller network tests used USB power.

Figure 9.1: The vibration platform used for testing accelerometers.

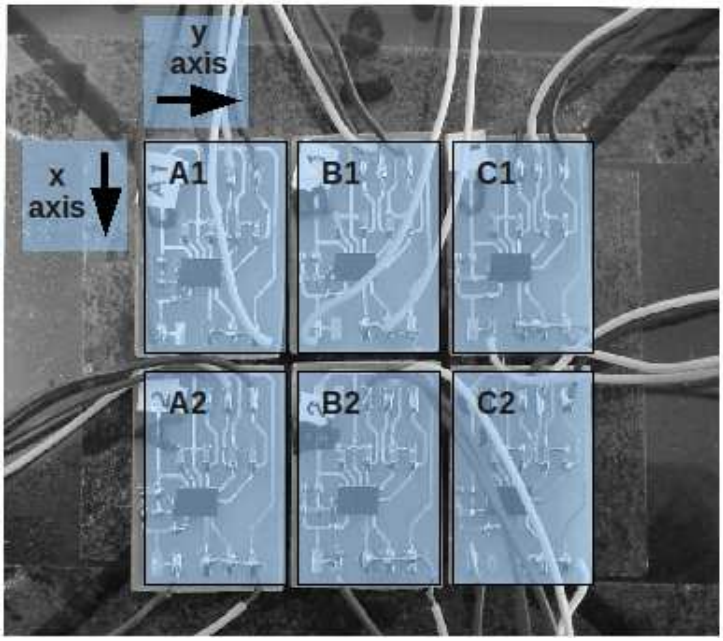


Figure 9.2: Positioning of sensors on vibration platform. The positions are shown in the same orientation as the platform in Figure 9.1.

Table 9.1: Comparison of amplitudes measured for accelerometer 1 on mote B.

	X axis	Y axis	Z axis
Oscilloscope			
Mean (mV)	24.16	32.31	521.60
σ (mV)	0.61	1.63	2.19
Mote B			
Channel	0 (x)	1 (y)	2 (z)
Mean (mV)	22.71	30.04	515.87
σ(mV)	1.43	1.18	0.86

9.1.1 Amplitude

The following experiments were performed to determine the accuracy of the amplitudes of the sampled signals.

The vibration platform, configured as described above, was fed a 100 Hz sine wave. The outputs of the accelerometers were sampled by the motes and logged to the computer. In addition, the outputs of one of the accelerometers were captured using the oscilloscope. Ten sets of values were collected and the mean and standard deviation calculated. Peak-to-peak voltages were obtained by using the peak-to-peak measurement function on the oscilloscope and by getting the minimum and maximum values of the sample set waveforms in MATLAB.

The results of the tests measuring the x, y and z axes of the first accelerometer on mote B are summarised in Table 9.1. The x, y and z axes were connected to ADC channels 0, 1 and 2, respectively. The motes all used a 2.5 V ADC reference. As such, the ADC values were converted to voltages with $V_{in} = N_{ADC} \times V_{R+}/4095$, where V_{R+} is the reference voltage, N_{ADC} is the value from the ADC and V_{in} is the input voltage [80].

It can be seen from the table that the values measured by the mote closely match those obtained from the oscilloscope. There is a greater discrepancy at very small amplitudes, as seen in the previous chapter.

9.1.2 Frequency

The vibration platform was controlled with a sinusoidal input. Data captured with the oscilloscope were compared with data captured by the motes. The experiment was repeated three times with an input frequency of 100 Hz, 300 Hz and 500 Hz. Figures 9.3, 9.4 and 9.5 show the 100 Hz, 300 Hz and 500 Hz signals from the z axis

for the oscilloscope and mote captures in the frequency domain. The data from the oscilloscope was sampled with a 20 μs sample period, and 2500 samples were provided. This created frequency divisions of 20 Hz.

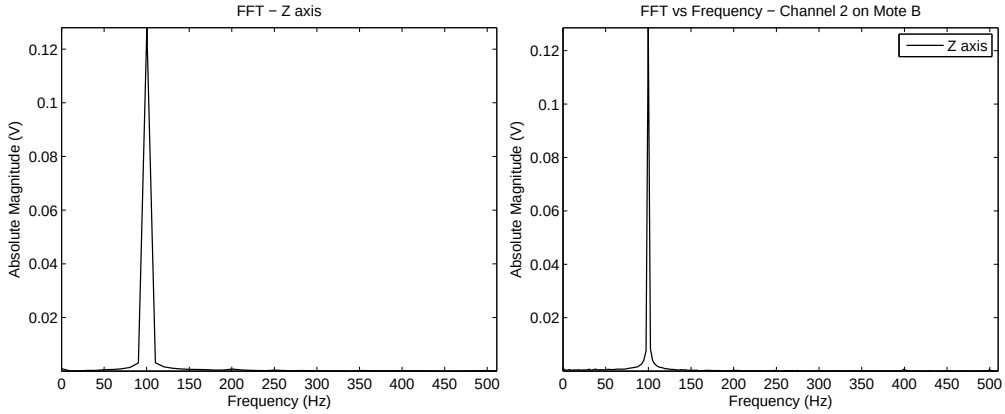


Figure 9.3: Mote B, z axis frequency spectrum plotted from oscilloscope data (left) and mote ADC data (right) with a 100 Hz sinusoidal input.

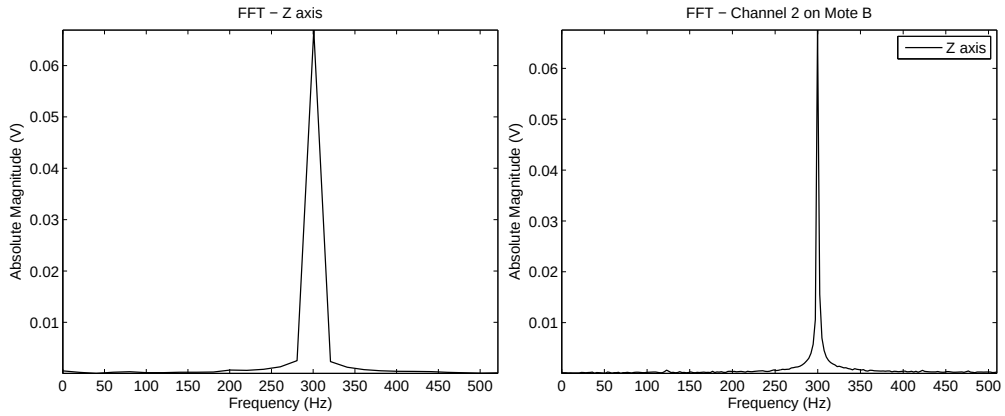


Figure 9.4: Mote B, z axis frequency spectrum plotted from oscilloscope data (left) and mote ADC data (right) with a 300 Hz sinusoidal input.

9.1.3 Synchronisation

By using the inter-channel delay measured in Section 8.6.1, the sampled waveform can be shown in the time domain. Figure 9.6 shows a portion of the waveform captured by the mote in the sample domain. The sensors were being subjected to motion caused by the platform being driven by a 100 Hz sine wave. Figure 9.7 shows the same portion of the sampled signal in the time domain using the known inter-channel delays. Imperfections in the sensor boards and noise on the input signal contribute to slight differences between the two channels.

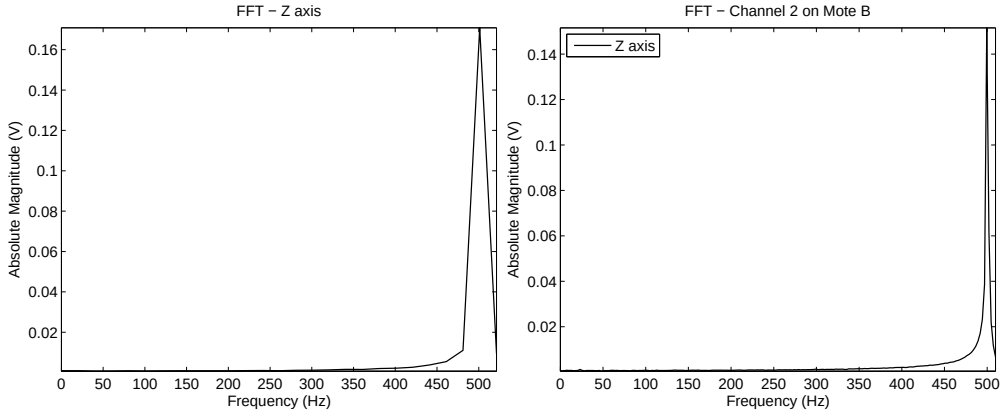


Figure 9.5: Mote B, z axis frequency spectrum plotted from oscilloscope data (left) and mote ADC data (right) with a 500 Hz sinusoidal input.

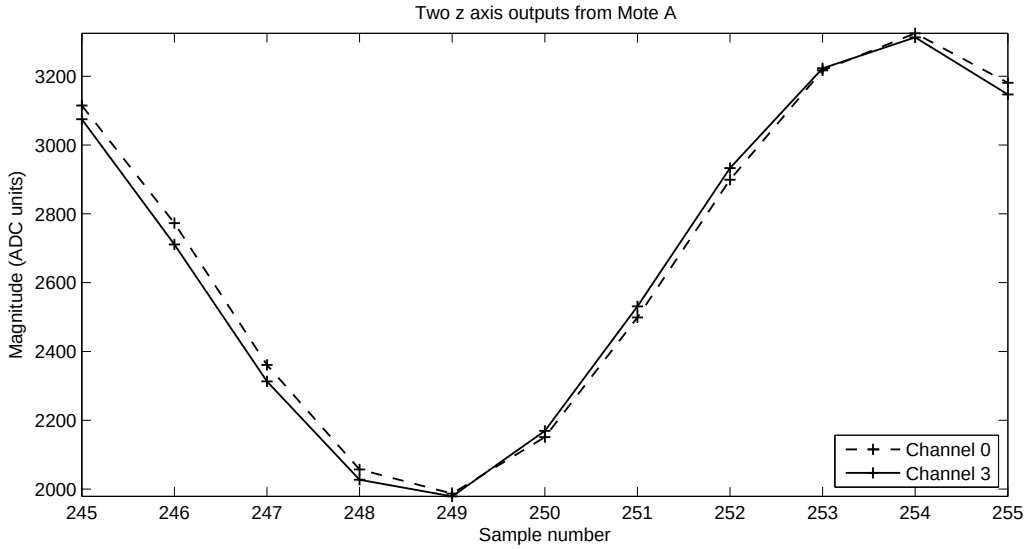


Figure 9.6: Z axis measurements from 2 accelerometers on mote A in response to a 100 Hz sine wave in the sample domain.

To illustrate the synchronisation between motes, the output from the z axis of each mote was plotted. The experiment matches that described in Section 8.6.2, with 4 motes out of sync with the others. Two of the motes were 21 clock cycles ahead of the majority, while 2 of the motes were 21 clock cycles behind the majority of motes. Figure 9.8 shows part of the sampled signal in the sample domain, while Figure 9.9 shows the same portion of the sampled signal in the time domain.

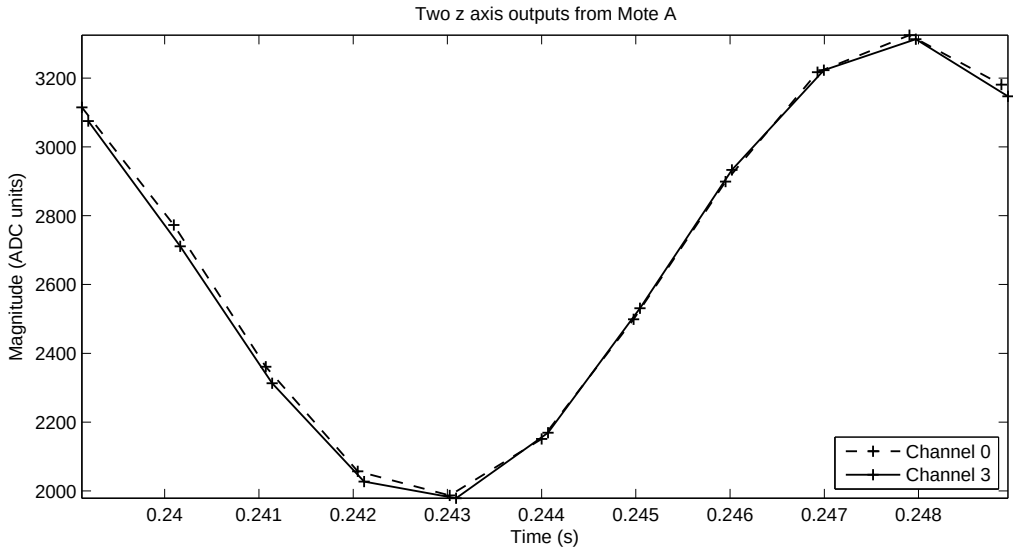


Figure 9.7: Z axis measurements from 2 accelerometers on mote A in response to a 100 Hz sine wave in the time domain.

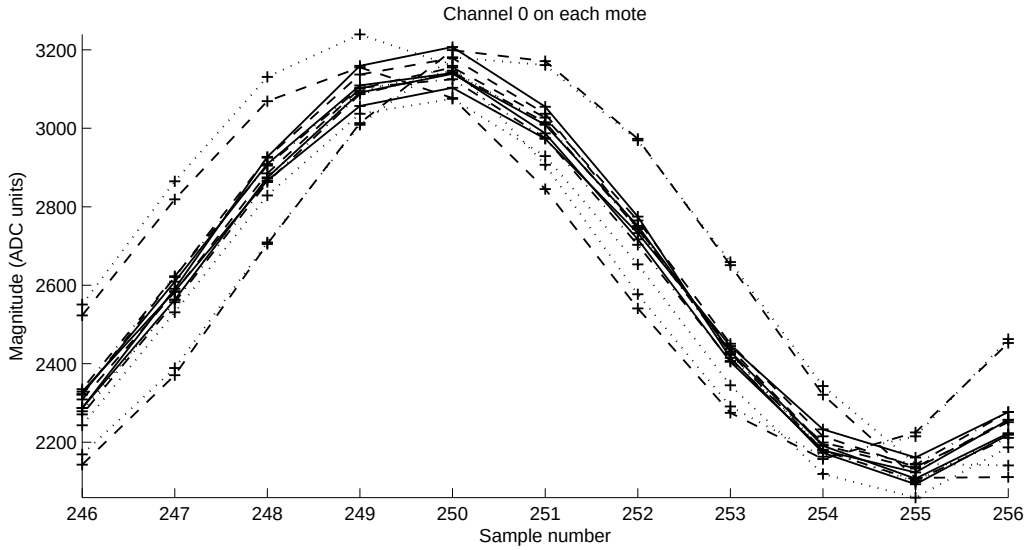


Figure 9.8: Z axis measurements from accelerometers on channel 0 across each mote in response to a 100 Hz sine wave in the sample domain.

9.2 Transformer application

The application of power transformer monitoring was proposed. As such it was necessary to run the system to measure the vibrations of a transformer. A 3 phase, 38 kVA / 47 kVA, 380 V / 220 V, 50 Hz, AC isolation transformer that was used to supply power to the heavy current laboratory on the CPUT Cape Town campus was used. This was conveniently located on campus and presented the opportunity to test various sensor configurations on the transformer in an indoor, fairly controlled environment. The transformer, shown in Figure 9.10, measured 720 mm by 960 mm

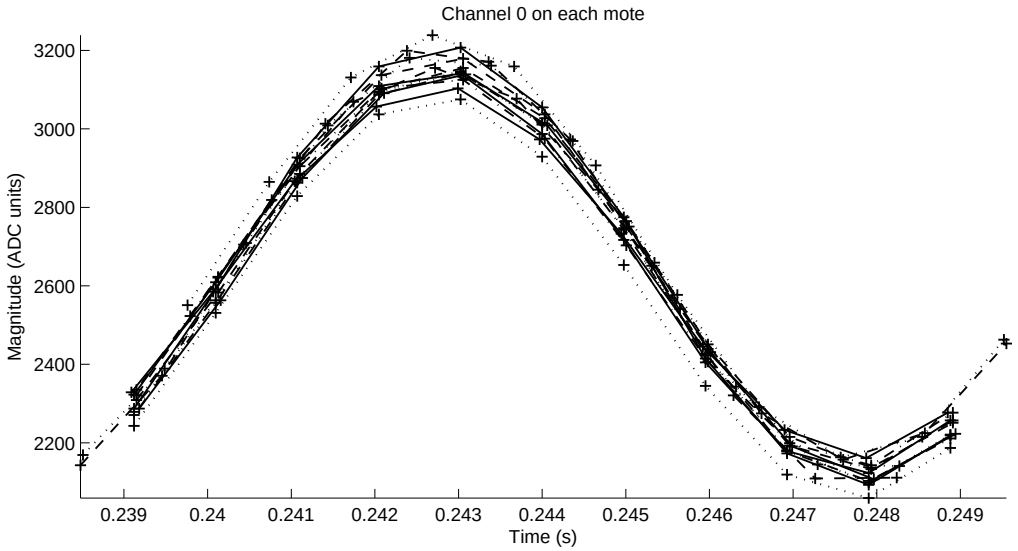


Figure 9.9: Z axis measurements from accelerometers on channel 0 across each mote in response to a 100 Hz sine wave transferred to the time domain.

by 335 mm with cooling pipes attached on the two larger sides. Measurements were taken on the unencumbered section of the transformer tank, between the cooling pipes. This area measured 720 mm by 560 mm.

A network was configured on the one side of the transformer using 15 sensor motes. The network measured vibrations perpendicular to the transformer surface at 15 points. The positions of the sensors are shown in Figure 9.11. The predominantly single axis sensors (some sampled 3 axes) were centred 144 mm apart horizontally and 186.7 mm apart vertically.

The sample and frequency domain plots from the centre single axis sensor are shown in Figure 9.12. There are strong 100 Hz and 200 Hz components. Magnetostriction in the core would be the dominant cause of the 100 Hz component [74].

Figure 9.13 shows plots representing the displacement of the tank surface from data sampled at each single-axis measured point for 100 Hz, 200 Hz, 300 Hz and 500 Hz. This was achieved by twice integrating the absolute magnitude data of each frequency of interest after converting the magnitude to acceleration in m/s^2. This was done in the frequency domain by dividing the magnitude by ω^2 for each chosen frequency. These values were inserted at the relevant co-ordinates of a zeroed grid representing the transformer tank surface. A Gaussian low pass filter was created and applied to the grid to smooth the surface to represent the movement of the rigid tank surface.

Figure 9.14 shows a comparison of 2 sensors in time. One sensor is at the far left of the transformer (closest to the wall in Figure 9.10 and the sensor is at the centre. It

Figure 9.10: Isolation transformer with 15 sensor motes.

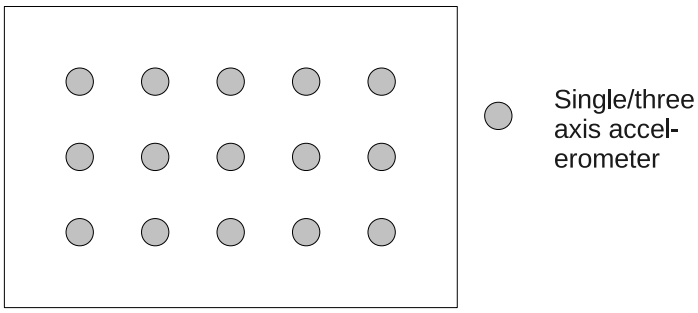


Figure 9.11: Positions of single and three axis sensors on transformer tank.

can be seen from the plot that the 100 Hz component of each signal are approximately 180° out of phase.

The bottom row of sensor motes was removed from the configuration described above and placed opposite the sensors of the centre row (from Figure 9.11). Figure 9.15 shows the central sensor on opposite sides in the time domain, while Figure 9.16 shows the captured data in the frequency domain.

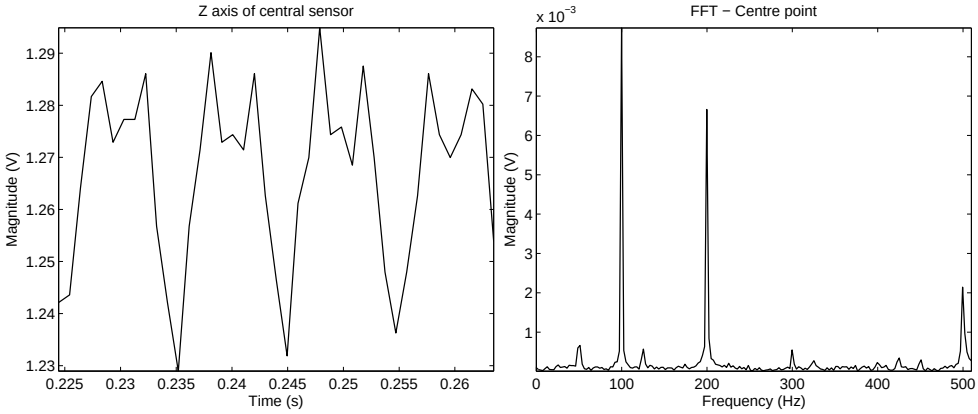
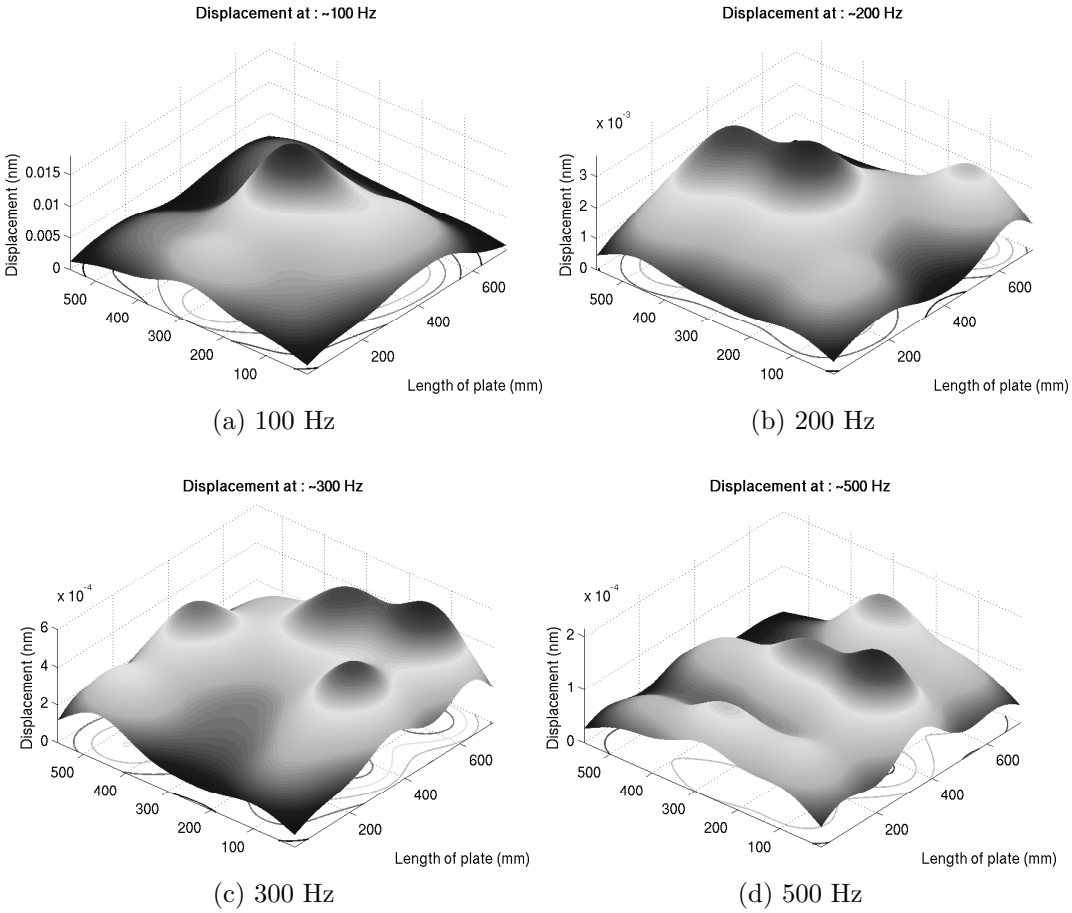


Figure 9.12: The time domain and frequency domain plots for the central sensor on the transformer.

9.3 Summary

A set of tests were run on the system with samples taken from attached accelerometers. The accelerometers were mounted on a vibration platform. Oscilloscope measurements from accelerometers closely match sampled values. Sensornets were deployed on an isolation transformer. While it is outside the scope of this project to interpret the results as related to the internal workings of the transformer, one can clearly see the 100 Hz component caused by magnetostriction in the core.



(a) 100 Hz (b) 200 Hz

(c) 300 Hz (d) 500 Hz

Figure 9.13: Plots of the displacement of the tank surface at approximately 100 Hz, 200 Hz, 300 Hz and 500 Hz.

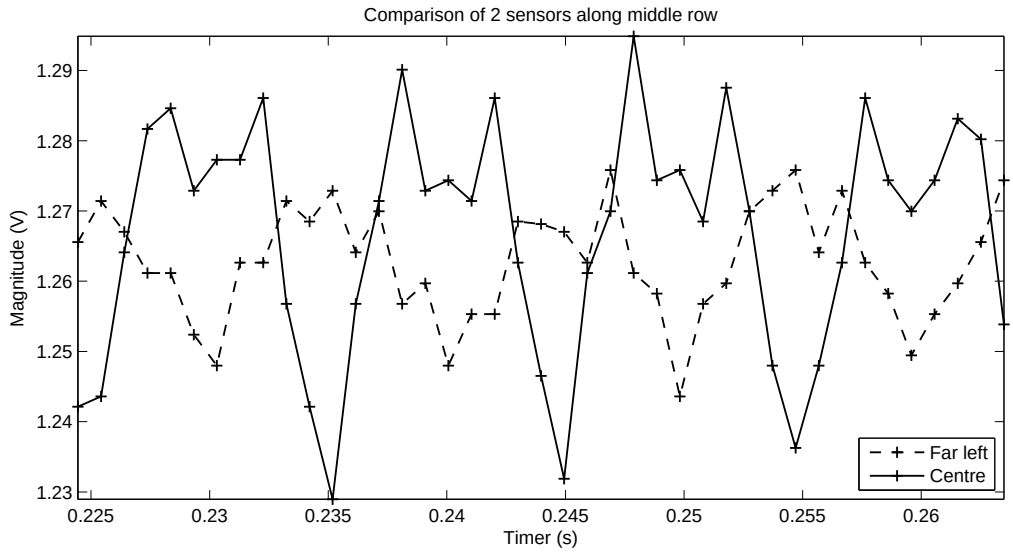


Figure 9.14: Measured values (with the mean removed) from 2 sensors on the transformer. The sensors are positioned along the middle row at the far left and in the centre.

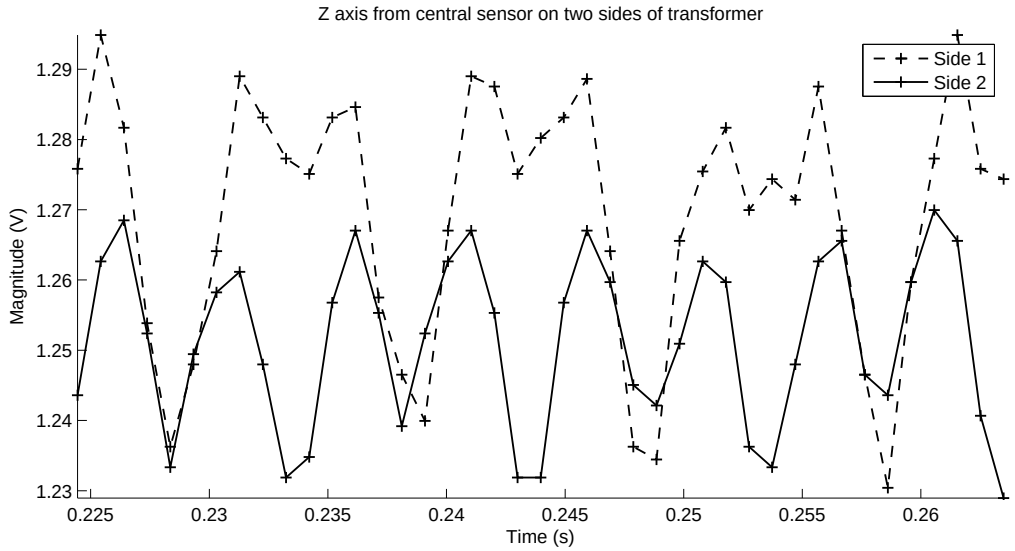


Figure 9.15: Time domain plot of 2 sensors centrally placed on the transformer tank opposite each other.

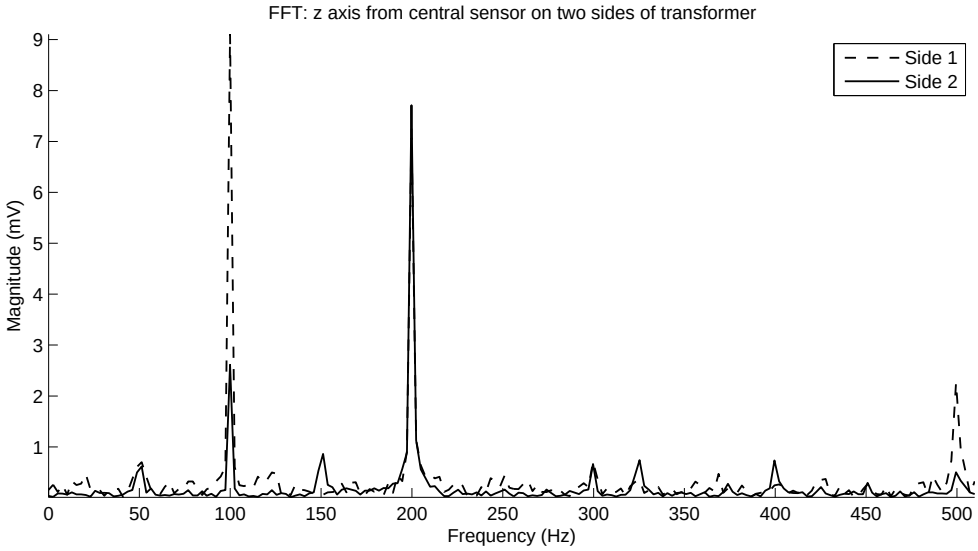


Figure 9.16: Frequency domain plot of 2 sensors centrally placed on the transformer tank opposite each other.

Chapter 10

Conclusions and future work

A sensornet for logging time synchronised data was designed and implemented in TinyOS on TelosB motes. The system comprised sensor motes to which accelerometers were attached and a base mote that controlled the system and forwarded collected data to a logging program. The sensor mote software was designed to be as simple as possible. The base mote configured the network and provided a global reference time to the sensor motes. The base mote also requested sampling to begin and tracked dropped data packets to provide a reliable data collection scheme. The amount of lost data considered recoverable was limited as the cost in energy to recover a large number of packets was high and empirical evidence showed that there was only very limited packet loss. The data logger wrote data and meta-data, currently only timestamps, to CSV files for later analysis. Sample sets with unrecovered data had files labeled, rather than outright discarded, in case there was a use for partial data sets.

The protocol was analysed and compared to simulated results. The protocol had a high data packet to control packet ratio. The simulations showed the reducing effect on packet loss and latency of staggering data packet transmission to reduce network interference. Empirically measured packet losses were higher than simulated measurements. This was expected as there were many small, low level timing and operation processes that were not included in the simulation that, though small, may have affected the results. Channel conditions and the presence of external interference may have contributed to the increase in losses. Further research into the matter is required.

Results show that measured signals were reproduced with high accuracy of frequency and amplitude, except at small amplitudes where the signal to noise ratio was poor. The mean delay between consecutive samples on a mote was 22.765 μs, close to the expected 21.8 μs calculated assuming a 5 MHz clock source. FTSP timestamps

enabled data from different motes to be synchronised in time, with an resolution of 30.5 μs. The sensor mote timestamps were typically within 1 to 2 clock cycles of each other. Phase comparisons between channels on different motes can be performed, but with limited accuracy due to the offsets.

The system was tested on an isolation transformer as per the proposed application of transformer monitoring. The measurements showed the dominant frequencies to be 100 Hz, 200 Hz, 300 Hz and 500 Hz. Further analysis is required to interpret these results.

10.1 Future work

If the output impedance of the accelerometers can be reduced, then the ADC sample time can be reduced which will result in shorter delays between consecutive samples. This will increase the accuracy of phase difference measurements. It will also help with sampling higher frequencies if necessary. A buffer on the output of the accelerometer will provide a low output impedance at the cost of additional hardware and a larger sensor board.

Empirical tests have shown that current sampling rate is close to the maximum sample rate the program will allow. If one were to consider higher sample rates, then the software will have to be carefully scrutinised to facilitate the increased sample rate.

The project was developed using off the shelf TelosB motes, however these motes have limitations for future research and are unsuited to field deployment. Most notably, only a limited set of pins are available at extension headers and the USB components add unnecessary cost and PCB space. A problem encountered during testing was, at times, poor connections at the expansion headers that forced the repeating of certain experiments. The issue of plugin connectors in the field was noted by Mainwaring et al. [55] during their Great Duck Island deployment. Creating a custom mote based on the Epic mote core should be considered for future development.

Though logging data to text files was convenient and appropriate for current testing, long term deployment would benefit from logging directly to a database.

The possibility of compressing data on the sensor motes prior to transmission should be investigated to reduce the amount of data sent and therefore reduce the power consumption.

Storing additional meta-data, such as supply voltage and temperature, from the sensor

motes may prove useful in calibrating data.

References

[1] A. E. B. Abu-Elanien and M. M. A. Salama, "Survey on the transformer condition monitoring," in *Large engineering systems conference on power engineering*. IEEE, Oct. 2007, pp. 187–191.

[2] O. Akan and I. Akyildiz, "Event-to-sink reliable transport in wireless sensor networks," *Networking, IEEE/ACM Transactions on*, vol. 13, no. 5, pp. 1003 – 1016, oct. 2005.

[3] I. F. Akyildiz, D. Pompili, and T. Melodia, "Underwater acoustic sensor networks: research challenges," *Ad Hoc Networks*, vol. 3, no. 3, pp. 257 – 279, 2005.

[4] Analog Devices, "ADXL335," Datasheet, 2009, rev. A. [Online]. Available: http://www.analog.com/en/sensors/inertial-sensors/adxl335/products/product.html

[5] Anon, "The network simulator - ns-2," [Last viewed: 24 August 2011]. [Online]. Available: http://isi.edu/nsnam/ns/

[6] C. Bartoletti, M. Desiderio, D. Di Carlo, G. Fazio, F. Muzi, G. Sacerdoti, and F. Salvatori, "Vibro-acoustic techniques to diagnose power transformers," *IEEE Transactions on Power Delivery*, vol. 19, no. 1, pp. 221–229, Jan. 2004.

[7] B. Berger, G. Blaylock, J. G. Charles, and A. B. Dobie, "Transformer noise," *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, vol. 263, no. 1142, pp. 381–411, Dec. 1968. [Online]. Available: http://www.jstor.org/stable/73398

[8] A. Boulis, *Castalia: A simulator for wireless sensor networks and body area networks*, 3rd ed., NICTA, Mar. 2011. [Online]. Available: http://castalia.npc.nicta.com.au/pdfs/Castalia-UserManual.pdf

[9] A. J. M. Cardoso and L. M. R. Oliveira, "Condition monitoring and diagnostics of power transformers," *International journal of COMADEM*, vol. 2, no. 3, pp. 5–11, Jul. 1999.

[10] M. Ceriotti, L. Mottola, G. P. Picco, A. L. Murphy, S. Guna, M. Corra, M. Pozzi, D. Zonta, and P. Zanon, "Monitoring heritage buildings with wireless sensor networks: The Torre Aquila deployment," in *IPSN '09: Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 277–288.

[11] Q. M. Chaudhari and E. Serpedin, "Clock estimation for long-term synchronization in wireless sensor networks with exponential delays," *EURASIP J. Adv. Signal Process*, vol. 2008, p. 27, 2008.

[12] K. Chintalapudi, J. Paek, O. Gnawali, T. S. Fu, K. Dantu, J. Caffrey, R. Govindan, E. Johnson, and S. Masri, "Structural damage detection and localization using NETSHM," in *IPSN '06: Proceedings of the 5th international conference on Information processing in sensor networks*. New York, NY, USA: ACM, 2006, pp. 475–482.

[13] C.-Y. Chong and S. P. Kumar, "Sensor networks: Evolution, opportunities, and challenges," *Proceedings of the IEEE*, vol. 91, no. 8, pp. 1247–1256, Aug. 2003.

[14] D. J. de Villiers, "Hybrid energy harvesting system for a condition monitoring mote," Master's thesis, Cape Peninsula University of Technology, Jun. 2009.

[15] D. J. de Villiers, S. Kaplan, and R. H. Wilkinson, "Energy harvesting for a condition monitoring mote," in *34th annual conference of IEEE Industrial Electronics*. IEEE, 2008, pp. 2161–2166.

[16] S. Deering and R. Hinden, "Internet Protocol, version 6 (IPv6) specification," RFC 2460, Dec. 1998.

[17] W. Dong, C. Chen, X. Liu, and J. Bu, "Providing os support for wireless sensor networks: Challenges and approaches," *Communications Surveys Tutorials, IEEE*, vol. PP, no. 99, pp. 1 –12, 2010.

[18] A. Dunkels, "The Contiki operating system," [Last viewed: 27 June 2010]. [Online]. Available: http://www.sics.se/contiki/

[19] A. Dunkels, "About Contiki," Feb. 2010, [Last viewed: 7 July 2010]. [Online]. Available: http://www.sics.se/contiki/about-contiki.html

[20] P. Dutta, J. Taneja, J. Jeong, X. Jiang, and D. Culler, "A building block approach to sensornet systems," in *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*. New York, NY, USA: ACM, 2008, pp. 267–280.

[21] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 147–163, 2002.

[22] R. Feinberg, Ed., *Modern power transformer practice*. Macmillan, 1979.

[23] R. Fonseca, O. Gnawali, K. Jamieson, S. Kim, P. Levis, and A. Woo, "The collection tree protocol (ctp)," TEP 123, Feb. 2007. [Online]. Available: http://www.tinyos.net/tinyos-2.x/doc/html/tep123.html

[24] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2003, pp. 138–149.

[25] B. Garcia, J. C. Burgos, and A. Alonso, "Winding deformations detection in power transformers by tank vibrations monitoring," *Electric power systems research*, vol. 74, pp. 129–138, Apr. 2005.

[26] J. A. Garcia-Souto and H. Lamela-Rivera, "Comparative analysis of optical-fibre interferometric sensors versus accelerometers: application to vibrations inside high-power transformers," *Journal of optics A: Pure and applied optics*, vol. 4, pp. S318–S326, Nov. 2002.

[27] D. Gay, "Oscilloscope.h," TinyOS source code file, 2006.

[28] D. Gay, P. Levis, D. Culler, and E. Brewer, "nesC 1.3 reference manual," Jul. 2009, [Last viewed: 22 June 2010]. [Online]. Available: http://nescc.cvs.sourceforge.net/viewvc/nescc/nesc/doc/ref.pdf?revision=1.19

[29] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '09. New York, NY, USA: ACM, 2009, pp. 1–14. [Online]. Available: http://doi.acm.org/10.1145/1644038.1644040

[30] V. Handziski, J. Polastre, J.-H. Hauer, C. Sharp, A. Wolisz, D. Culler, and D. Gay, "Hardware abstraction architecture," TEP 2. [Online]. Available: http://www.tinyos.net/tinyos-2.1.0/doc/html/tep2.html

[31] W. R. Heinzelman, J. Kulik, and H. Balakrishnan, "Adaptive protocols for information dissemination in wireless sensor networks," in *Fifth ACM/IEEE MOBICOM Conference, Seattle, WA*. ACM, 1999.

[32] J. Hill, "System architecture for wireless sensor networks," Ph.D. dissertation, Universtity of California, Berkeley, 2003.

[33] Y. S. Hong and J. H. No, "Time synchronization in wireless sensor network applications," in *SEUS'07: Proceedings of the 5th IFIP WG 10.2 international conference on Software technologies for embedded and ubiquitous systems*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 429–435.

[34] D.-J. Huang, W.-C. Teng, C.-Y. Wang, H.-Y. Huang, and J. Hellerstein, "Clock skew based node identification in wireless sensor networks," in *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, nov. 2008, pp. 1 –5.

[35] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, ser. SenSys '04. New York, NY, USA: ACM, 2004, pp. 81–94. [Online]. Available: http://doi.acm.org/10.1145/1031495.1031506

[36] IEEE, "IEEE standard for information technology – telecommunication and information exchange between systems – local and metropolitan area networks – specific requirements. part 15.4: Wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (LR-WPANs)," IEEE Computer Society, New York, NY, USA, Oct. 2003.

[37] IEEE, "IEEE standard for information technology- telecommunications and information exchange between systems- local and metropolitan area networks-specific requirements part 15.4: Wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (WPANs)," Sep. 2006, revision of 802.15.4-2003.

[38] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," in *Proceedings of the 6th annual international conference on Mobile computing and networking*, ser. MobiCom '00. New York, NY, USA: ACM, 2000, pp. 56–67. [Online]. Available: http://doi.acm.org/10.1145/345910.345920

[39] P. Kang, D. Birtwhistle, J. Daley, and D. McCulloch, "Noninvasive on-line condition monitoring of on load tap changers," in *Power Engineering Society Winter Meeting, 2000. IEEE*, vol. 3, 23-27 2000, pp. 2223 –2228 vol.3.

[40] S. Kaplan, D. de Villiers, L. Steenkamp, G. de Jager, J. Davies, and R. Wilkinson, "Towards power transformer condition monitoring," in *SenSys '09: Proceedings*

of the 7th ACM Conference on Embedded Networked Sensor Systems. New York, NY, USA: ACM, 2009, pp. 323–324.

[41] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon, "Health monitoring of civil infrastructures using wireless sensor networks," in *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*. New York, NY, USA: ACM, 2007, pp. 254–263.

[42] J. D. Kleynhans, "Remote vibration and magnetic field analysis of power transformers," BTech thesis, Nov. 2008, Cape Peninsula University of Technology.

[43] K. Klues, C.-J. M. Liang, J. Paek, R. Musăloiu-E, P. Levis, A. Terzis, and R. Govindan, "Tosthreads: thread-safe and non-invasive preemption in tinyos," in *SenSys '09: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*. New York, NY, USA: ACM, 2009, pp. 127–140.

[44] K. Klues, C.-J. M. Liang, J. Paek, R. Musăloiu-E, A. Terzis, R. Govindan, and P. Levis, "The TOSThreads thread library," TEP 134, May 2008. [Online]. Available: http://www.tinyos.net/tinyos-2.1.0/doc/html/tep134.html

[45] R. Kumar and S. Ganeriwal, "TPSNsyncM.nc," 2003. [Online]. Available: http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-1.x/contrib/Timesync-NESL-UCLA/tos/system/

[46] S. H. Lee and L. Choi, "Chaining clock synchronization: An energy-efficient clock synchronization scheme for wireless sensor networks," in *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on*, 14-16 2009, pp. 172 –177.

[47] P. Levis and D. Gay, *TinyOS programming*, 1st ed. New York: Cambridge University Press, Apr. 2009.

[48] P. Levis, D. Gay, V. Handziski, J.-H. Hauer, B. Greenstein, M. Turon, J. Huio, K. Klues, C. Sharp, R. Szewczyk, J. Polastre, P. Buonadonnao, L. Nachman, G. Tolleo, D. Cullero, and A. Wolisz, "T2: A second generation OS for embedded sensor networks," TKN-05-007. [Online]. Available: http://sing.stanford.edu/pubs/tinyos2-tr.pdf

[49] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: Accurate and scalable simulation of entire tinyos applications," in *SensSys 2003*. ACM, 2003.

[50] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *Proceedings of the 1st conference on Symposium on Networked Systems Design*

and Implementation - Volume 1. Berkeley, CA, USA: USENIX Association, 2004, pp. 2–2. [Online]. Available: http://portal.acm.org/citation.cfm?id=1251175.1251177

[51] P. Levis and G. Tolle, "Dissemination of small values," TEP 118.

[52] C.-J. M. Liang, J. Liu, L. Luo, A. Terzis, and F. Zhao, "Racnet: a high-fidelity data center sensing network," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '09. New York, NY, USA: ACM, 2009, pp. 15–28. [Online]. Available: http://doi.acm.org/10.1145/1644038.1644041

[53] Libelium, "Waspmote datasheet," Jun. 2010, version 0.5. [Online]. Available: http://www.libelium.com/support/waspmote

[54] K. Lin and P. Levis, "Data discovery and dissemination with dip," in *Information Processing in Sensor Networks, 2008. IPSN '08. International Conference on*, april 2008, pp. 433 –444.

[55] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*. New York, NY, USA: ACM, Sep. 2002, pp. 88–97.

[56] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "The flooding time synchronization protocol," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2004, pp. 39–49.

[57] M. Maróti and B. Kusy, "TimeSyncMode.nc," TinyOS source code file, 2002, ported to TinyOS2 by Brano Kusy, 2008.

[58] M. Maróti and B. Kusy, "TimeSyncMsg.h," TinyOS source code file, 2002, ported to TinyOS2 by Brano Kusy, 2008.

[59] R. S. Masti, W. Desmet, and W. Heylen, "On the influence of core laminations upon power transformer noise," in *Proceedings of ISMA*, 2004, pp. 3851–3862.

[60] Memsic, "Imote2: high-performance wireless sensor network node," 6020-0117-04 Rev A. [Online]. Available: http://www.memsic.com/products/wireless-sensor-networks/wireless-modules.html

[61] Memsic, "MICAz: wireless measurement system," 6020-0065-05 Rev A. [Online]. Available: http://www.memsic.com/products/wireless-sensor-networks/wireless-modules.html

[62] Memsic Corporation, "Wireless modules," 2010, [Last viewed: 22 June 2010]. [Online]. Available: http://www.memsic.com/products/wireless-sensor-networks/wireless-modules.html

[63] D. Mills, "Simple Network Time Protocol (SNTP) version 4 for IPv4, IPv6 and OSI," RFC 2030, Oct. 1996.

[64] D. Mills, J. Martin, J. Burbank, and W. Kasch, "Network Time Protocol version 4: Protocol and algorithms specification," RFC 5905, Jun. 2010.

[65] D. Mills, "Network Time Protocol (version 3) specification, implementation and analysis," RFC 1305, Mar. 1992.

[66] NICTA, "Castalia home," [Last viewed: 15 August 2011]. [Online]. Available: http://castalia.npc.nicta.com.au/

[67] P. Nirgude, B. Gunasekaran, Channakeshava, A. Rajkumar, and B. Singh, "Frequency response analysis approach for condition monitoring of transformer," in *Conference on electrical insulation and dielectric phenomena*, Oct. 2004, pp. 186–189.

[68] OMNeT++ Community, "OMNeT++ network simulation framework," [Last viewed: 24 August 2011]. [Online]. Available: http://www.omnetpp.org/

[69] J. Paek and R. Govindan, "Rcrt: Rate-controlled reliable transport protocol for wireless sensor networks," *ACM Trans. Sen. Netw.*, vol. 7, pp. 20:1–20:45, October 2010. [Online]. Available: http://doi.acm.org/10.1145/1807048.1807049

[70] J. Polastre, R. Szewczyk, and D. Culler, "Telos: enabling ultra-low power wireless research," in *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*. Piscataway, NJ, USA: IEEE Press, 2005, p. 48.

[71] S. Rangwala, R. Gummadi, R. Govindan, and K. Psounis, "Interference-aware fair rate control in wireless sensor networks," in *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '06. New York, NY, USA: ACM, 2006, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1159913.1159922

[72] V. Rusov and S. Zhivodernikov, "Transformer condition monitoring," in *International Conference on Condition Monitoring and Diagnosis, 2008. CMD 2008*, Apr. 2008, pp. 1012–1014.

[73] C. Sharp, M. Turon, and D. Gay, "Timers," TEP 102. [Online]. Available: http://www.tinyos.net/tinyos-2.1.0/doc/html/tep102.html

[74] J. Shengchang, L. Yongfen, and L. Yanming, "Research on extraction technique of transformer core fundamental frequency vibration based on OLCM," *IEEE Transactions on Power Delivery*, vol. 21, pp. 1981–1988, Oct. 2006.

[75] W.-Z. Song, R. Huang, M. Xu, A. Ma, B. Shirazi, and R. LaHusen, "Air-dropped sensor network for real-time high-fidelity volcano monitoring," in *MobiSys '09: Proceedings of the 7th international conference on Mobile systems, applications, and services*. New York, NY, USA: ACM, 2009, pp. 305–318.

[76] F. Stann and J. Heidemann, "Rmst: reliable data transport in sensor networks," in *Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on*, may 2003, pp. 102 – 112.

[77] STMicroelectronics, "LIS344ALH," Datasheet, Apr. 2008. [Online]. Available: http://www.st.com/stonline/products/families/sensors/motion_sensors/lis344alh.htm

[78] STMicroelectronics, "LIS352AX," Mar. 2009. [Online]. Available: http://www.st.com/stonline/products/literature/ds/15530/lis352ax.htm

[79] Sun Microsystems, "SunSPOTWorld - home," [Last viewed: 27 June 2010]. [Online]. Available: http://sunspotworld.com/index.html

[80] Texas Instruments, *MSP430x1xx family user's guide*, 2006, rev. F.

[81] Texas Instruments, *CC2420: 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver*, SWRS041b ed., Mar. 2007. [Online]. Available: http://focus.ti.com/docs/prod/folders/print/cc2420.html

[82] Texas Instruments, "MSP430F15x, MSP430F16x, MSP430F161x mixed signal microcontroller," Data sheet, May 2009.

[83] G. Tolle and D. Culler, "Design of an application-cooperative management system for wireless sensor networks," in *Wireless Sensor Networks, 2005. Proceeedings of the Second European Workshop on*, jan.-2 feb. 2005, pp. 121 – 132.

[84] UC Berkeley, "TinyOS community forum —— an open-source OS for the networked sensor regime," [Last viewed: 27 June 2010]. [Online]. Available: http://www.tinyos.net

[85] UC Berkeley, "Telos Rev. B," Schematic, Sep. 2004. [Online]. Available: http://webs.cs.berkeley.edu/tos/hardware/telos/telos-revb-2004-09-27.pdf

[86] UC Berkeley, "TinyOS community forum —— hardware designs," 2004, [Last visited: 27 June 2010]. [Online]. Available: http://www.tinyos.net/scoop/special/hardware

[87] C.-Y. Wan, A. Campbell, and L. Krishnamurthy, "Pump-slowly, fetch-quickly (psfq): a reliable transport protocol for sensor networks," *Selected Areas in Communications, IEEE Journal on*, vol. 23, no. 4, pp. 862 – 872, april 2005.

[88] C.-Y. Wan, S. B. Eisenman, and A. T. Campbell, "Coda: congestion detection and avoidance in sensor networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, ser. SenSys '03. New York, NY, USA: ACM, 2003, pp. 266–279. [Online]. Available: http://doi.acm.org/10.1145/958491.958523

[89] B. Warneke, M. Last, B. Liebowitz, and K. Pister, "Smart dust: communicating with a cubic-millimeter computer," *Computer*, vol. 34, no. 1, pp. 44 –51, jan 2001.

[90] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh, "Fidelity and yield in a volcano monitoring sensor network," in *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation.* Berkeley, CA, USA: USENIX Association, 2006, pp. 381–396.

[91] G. Werner-Allen, S. Dawson-Haggerty, and M. Welsh, "Lance: optimizing high-resolution signal collection in wireless sensor networks," in *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems.* New York, NY, USA: ACM, 2008, pp. 169–182.

Appendix A

Computer source code