"Evaluating Wireless Sensor Network Quality of Service Using the iHEED Application"

Honours Final Project Report

By

Student L 09-10

2006xxxxx

Project Supervisor: Professor Huaglory Tianfield

2nd Marker: Dr. Richard Foley

"Except where explicitly stated all work in this document is my own."

Signed:_____ Date_____

Abstract

Due to their small size and inexpensive price, wireless sensor devices have a wide variety of uses and applications ranging from environment monitoring, to object tracking in areas as diverse as scientific research and armed forces battlefield options. Because most of these devices are battery powered, and thus subject to tight energy constraints, a great deal of research has focused on conserving energy. However as applications for wireless sensor networks become increasingly complex and demanding, many researchers are beginning to look at Quality of Service as an area applicable to wireless sensor networks.

Due to the unique nature of wireless sensor devices, it is unlikely that traditional QoS methods which rely on complicated marking schemes or resource allocation will be suitable for wireless sensor networks. This has lead to many researchers seeking alternative means of providing QoS to wireless sensor networks. Two leading approaches to doing so are clustering and data aggregation.

iHEED is an application that combines clustering and data aggregation and runs on the popular TinyOS platform. This project will focus on adapting iHEED to allow for its simulation in order to determine what effect clustering and data aggregation can have on a wireless sensor network's Quality of Service. Although iHEED has been run on actual nodes, it has not been simulated, and so at the time of writing this project is unique in attempting to adapt and simulate the iHEED application.

Acknowledgements

I would like to thank my supervisor Professor Huaglory Tianfield for his patience and guidance throughout the duration of this project.

I would like to thank my Mum, my Dad, and my sister Lynsey for their support, proof-reading, and general patience. I would especially like to thank my Dad in helping me get this bound.

I would also like to thank Carl, Grant, and Graeme for their assistance and for helping me to take a more practical approach.

Finally I would like to thank Natalie and Scott for their friendship and support.

Contents

Introduction

Background

The Significance of Wireless Sensor Networks

"It is not unreasonable to expect that in 10-15 years that the world will be covered with wireless sensor networks with access to them via the Internet" (Stankovic, 2006).

Wireless Sensor Networks (WSNs) typically consist of a group of several sensor nodes which are small devices containing an antenna, a transceiver, memory, a CPU, some sort of sensing device and a power source (Stavrou, 2005). As the cost of manufacturing such devices decreases and the processing power increases, the usage of and potential applications for these devices will likely increase (Taleghan *et al*, 2007).

Potential applications for wireless sensor networks cover a diverse range of areas from monitoring of physical environments to military applications (Chen and Varshney, 2004). Rommer and Martin (2004) go so far as to state that because wireless sensor devices have so many current and potential applications it can be difficult to determine the exact nature of wireless sensor networks.

An important use of WSN technology is proposed by Frye (2007) who explores the potential use of Wireless Sensor Networks by fire departments dealing with emergency situations. Enel and Martinet (2005) have published a paper examining the potential use of WSNs in naval combat systems.

WSNs are also of use to scientists and other researchers interested in gathering data about an environment, for instance, the system being designed by Peterson *et al* (2009) is intended to be used as part of an experiment in monitoring Mt. St. Helens.

An interesting area of research is that of wireless sensor/actuator networks (WSAN's) in which sensor devices combined are combined with actuators – electronic devices which can control

their physical environment (Xia, 2008). Xia *et al* (2008) propose a radical vision in which WSANs are embedded in everyday objects throughout the world, sensing, controlling, and communicating with each other and ordinary objects.

The general idea of embedding sensors in everyday objects goes back to the work of Mark Weiser, who pioneered the concept of ubiquitous computing in which computational devices were embedded into everyday objects (Weiser, 1993). However, while some such as Hill *et al* (2004) argue that wireless sensor networks are a vital step in realising the goal of ubiquitous computing, others such as Langendoen and Reijers (2003) argue that wireless sensor networks are currently unsuitable for use in ubiquitous scenarios and need to be treated separately.

Quality of Service in Wireless Sensor Networks

In traditional, wired, networks QoS demands are usually based on bandwidth consumption and are met through resource and traffic control which aren't necessarily appropriate for wireless sensor networks (Chen *et al*, 2004). Similarly, Younis (Younis *et al*, 2004), argues that conventional wireless network QoS is unsuitable for wireless sensor networks due to the specific constraints that wireless sensor networks are subject to. Therefore, it would seem, that in order to provide QoS in wireless sensor networks, it is necessary to develop more specific means of achieving this.

Before these means can be developed, however, it is important that the reasons that prevent traditional wired and wireless QoS-handling techniques being applied to wireless sensor networks are taken into account.

Younis *et al* (Younis *et al*, 2004), lists the following as WSN-specific QoS challenges; “bandwidth limitation, removal of redundancy, energy and delay trade-off, buffer size limitation, and support of multiple traffic types”. While Chen *et al* (Chen *et al*, 2004) identifies “severe resource constraints, unbalanced traffic, data redundancy, network dynamics, energy balance, scalability, multiple sinks, multiple traffic types, and packet criticality” as potential QoS issues.

From the two quotations above, it is clear that most of these issues more or less relate to the physical nature of the actual sensors. However, from a networking point of view, there are still

developments and optimisations that can be made in order to improve QoS in wireless sensor networks.

The iHEED Application

The goal of this project is to investigate the effect that clustering and data aggregation have on Quality of Service. To perform this test, the iHEED application will be implemented and tested. IHEED is an application designed for the TinyOS operating system, its purpose is to provide clustering and data aggregation functions to sensor nodes running the TinyOS operating system (Younis and Fahmy, 2005).

iHEED is named after the clustering protocol that it uses – HEED or Hybrid Energy-Efficient Clustering protocol (Younis and Fahmy, 2004). Clustering is a concept in which nodes split themselves into groups (known as clusters) in order to overcome some of the difficulties facing wireless sensor networks (i.e. limited radio range) and organise the network (Tang and Li, 2006). In some clustering schemes (such as HEED) there will be a node which acts as the cluster-head and is responsible for receiving traffic from member nodes and forwarding it onto the base (Younis and Fahmy, 2004).

Another function that is sometimes provided by cluster-heads, and that is featured in iHEED is data aggregation, which in clustered systems is where a cluster-head combines all of the packets it receives from cluster members before sending the combined packet to the base node (Mhatre and Rosenborg, 2004). The purpose of this is to reduce the number of transmissions that the cluster-head needs to make, as well as cutting down on network traffic, thus achieving the aims of reducing energy consumption and congestion (Mhatre and Rosenborg, 2004).

Project Objectives and Research Question

The overall aim of this project is to investigate the use of clustering and data aggregation in WSN's from a QoS perspective. More specifically, this project is an experimentally based project in which a specific application that implements both clustering and data aggregation (iHEED) will be simulated in order to determine whether or not the clustering and data aggregation has provided an improved level of QoS. The application that iHEED will be compared against is the

Surge application of which iHEED is a modification/augmentation of (Younis and Fahmy, 2005).

Although this application has been evaluated by its authors, at the time of writing, it has not yet undergone any significant independent analysis, and nor has it been simulated. Indeed, original attempts at implementation found the application needed modification in order to be simulated. Thus in addition to performing a rigorous, independent, analysis of the protocol from a QoS perspective, another major contribution of this work is the modification of the iHEED application to allow for simulation using the TOSSIM simulator.

Before any of the actual experimentation or simulation can begin, it is important that an in-depth literature review is performed in order to gain a broad understanding of areas related to WSN's as well as a more detailed knowledge of critical aspects such as QoS and a technical understanding of the implementation platform (TinyOS). From this literature review, a detailed implementation, testing, and evaluation strategy can be prepared to ensure that the actual experimentation phase is as productive as possible.

In order to drive both individual aspects (i.e. the literature review, experiments, etc.) as well as the project as a whole, the following research question was developed.

Research Question

“How effective is the iHEED application’s combination of clustering and data aggregation at improving Quality of Service in a TinyOS-based Wireless Sensor Network?”

Unlike the project title, which is more general, the aim of the research question is to specifically state both the goal of the project (investigating the impact of data aggregation and clustering on WSN QoS) as well as the context (the iHEED application running on TinyOS-based nodes), however it is still sufficiently high-level enough to be used as the general basis for determining more specific aims and objectives.

This is the very purpose of the next sub-section which takes the research question, and uses it as the basis for developing several project objectives that represent key phases and activities that

must be undertaken/accomplished in order to answer the research question, thus fulfilling the aims of the project.

Project Objectives

As the previous sub-section noted, the research question is a more focussed version of the project title, yet is still somewhat high-level and general, albeit more specific than the research question it was derived from. From this comparatively high-level and general research question the following project objectives have been derived, yet like the project title and research question, these objectives will themselves be the genesis for more detailed and lower-level objectives in the form of the literature review and primary research objectives that will be discussed later on.

With this in mind, the following objectives have drawn up based on both the research question and project title that proceeds them, these are somewhat similar to the objectives presented in the interim report, however this is because the general aim of the project has changed very little during the period between the submission of the interim report and the formal commencement of the research stage.

Objective One: Research Wireless Sensor Networking

The purpose of this objective is to build on earlier secondary research in order to develop a better understanding of what wireless sensor networks are, what they are used for, and how they work. Although these might actually sound like rather general topics, it is important that these basic concepts are fully understood before more advanced topics are considered.

The purpose of this objective is not just to obtain knowledge for the sake of obtaining knowledge, but to develop the necessary background for further secondary research in more specific areas, as well as acting as preparation for the practical research stage.

Objective Two: Understand WSN QoS

After the previous objective has been completed, the focus becomes increasingly narrower, moving away from wireless sensor networks in general, and focussing on Quality of Service as a particular area of interest. QoS can be a somewhat ambiguous term, subject to various definitions depending on the context, therefore this part of the project is concerned with researching and reviewing various approaches QoS.

Of course, the focus here is not QoS in general, but QoS in the context of wireless sensor networks, and so it is important that emphasis is placed on evaluating these approaches with regards to wireless sensor networks.

In addition to looking at approaches to QoS, it is also important to attempt to understand how QoS is defined, not just in terms of implementation, but in terms of measurement. Therefore another sub-objective here would be to examine the various definitions of QoS, again evaluating their relevance to wireless sensor networks.

Objective Three: Research Clustering and Data Aggregation Techniques

Upon completion of the last objective, a suitably detailed knowledge of wireless sensor network QoS should have been gained, and from this suitable QoS parameters and metrics derived. Once QoS has been defined it is important to focus on how the desired level of QoS can be achieved. This objective will begin with an attempt to gain a more detailed knowledge of WSN clustering and aggregation techniques before looking at the HEED protocol in more detail. The first part will therefore involve a review of various sources of literature related to clustering and data aggregation algorithms; from surveys of multiple algorithms to more specific papers introducing a particular algorithm.

The aim here is to review multiple clustering and data aggregation algorithms and attempt to discover both their similarities and differences. Initially this will be a high-level review, focussing on increasing knowledge of clustering and data aggregation in general, however, as the review progresses, more emphasis will be placed on evaluating these algorithms from a QoS perspective. While some algorithms will explicitly deal with QoS, others will not mention it, but that does not mean that they will achieve some form of QoS.

Objective Four: Research HEED and iHEED

Once the review of clustering and data aggregation methods has been completed, attention will turn to the HEED protocol as this clustering algorithm and the accompanying data aggregation approach found in the iHEED application will form the basis of the primary research stage.

Using the general knowledge gained from the previous stage, the HEED protocol will be reviewed both from a general clustering perspective as well as from a more detailed QoS perspective. A similar process will occur with the data aggregation element of the iHEED application, although unfortunately less information is available on it than with HEED.

With the theoretical aspects of HEED and data aggregation fully understood, emphasis will move to the actual iHEED application, which is this project's test subject, so to speak. This part of the project will include analytical activities such as reviewing documentation and source code in order to transition the project from the literature review stage to the primary research stage which will be covered by the next two objectives.

Objective Five: Design the Testing and Evaluation Scheme

The primary goal of the last objective was to gain both a theoretical and working knowledge of the HEED protocol both from a general clustering and more specific QoS perspective. Using this knowledge a basic testing plan can be composed. Although it is not intended that full working knowledge of the implementation and simulation environments will have been acquired at this stage, it is still necessary to design and finalised the testing plan before implementation begins.

Designing the test plan at this stage means that it can be drawn up based on the overall aims and objectives of this project rather than based on lower level technical goals. That is to say, that the testing plan is the logical conclusion of the literature review process as it uses the knowledge accumulated in previous stages (i.e. understanding of clustering and QoS) and uses this as the basis for the testing and evaluation scheme.

As this stage will involve the formal identification of metrics it is linked to the second objective in which QoS metrics are researched and evaluated based on their suitability for use in wireless sensor networking. Although it is expected that some conclusions will be drawn at the end of the

second objective, because the focus of the project at that stage is still on the theoretical rather than practical aspects, it is not expected that concrete metrics will be drawn up.

Instead, having already gained sufficient theoretical knowledge of wireless sensor networks, QoS, and the particular technologies being examined in this project as well as some basic practical knowledge of TinyOS and the iHEED application this stage is a more appropriate point at which to determine what metrics will be used.

Objective Six: Implement and Test the iHEED Application

While the last objective could be seen as the bridge between the secondary and primary research stages, this objective is firmly rooted in the primary research stage and is almost entirely practical. The first part of this objective involves implementing the environment in which the HEED protocol will be tested and evaluated. This involves altering TinyOS to run the iHEED application and then carrying out preliminary testing to ensure that the application has been implemented correctly and is functioning as it should.

Following on from this preliminary testing, work will begin on translating the testing and evaluation scheme devised in the previous object into a practical simulation scenario. Using the testing scheme and the overall aims of the project as a guide, TinyOS's TOSSIM simulator will be configured so as to simulate a realistic application scenario from which relevant metrics (i.e. those QoS metrics defined in Objective Five) will be captured.

Objective Seven: Evaluate the Test Results and Draw Conclusions on the Effectiveness of iHEED:

With the iHEED application implemented and simulation completed, the focus of the primary research turns from testing to evaluation. As with the testing plan, the evaluation of these results will be based on the work of Objective Four in which knowledge acquired from previous stages (especially Objectives 3 and 4) will be used to analyse the test data and determine how effective the HEED protocol has been in coping with QoS demands. Based on the evaluation, possible modifications and topics for future research will then be proposed.

It should be noted that the majority of these objectives fall into the category of either literature review or primary research objectives, although objective four is intended as a bridge between these two sections. In the interim report separate, more detailed literature review and primary research were drawn up using these objectives as a guide. The reason that separate literature review objectives haven't been drawn up here is that because it is felt that the objectives given above are sufficiently detailed and focussed in order to drive the literature review without the need for further objectives to be drawn up.

Literature Review

As the previous section stated, there are no separate objectives for the literature report as it was decided that the project objectives were suitably detailed. One important point to not is that where the interim report's literature review was structured in almost exactly the same sequence as the objectives were presented this is not the case with the literature review here, as for instance, the definition of QoS section comes after the sections exploring clustering, data aggregation, and iHEED.

The reason for this change is that it allows for the general concepts of QoS to be explained before going on to examine different technologies used to implement it, and then more specifically looking at iHEED and the technologies behind it. This means that by the time the reader gets to the section defining QoS, they will have already read a significant amount of detail about the iHEED application as well as QoS in general, and it is hoped they will then be able to see why the particular metrics that were chosen to evaluate iHEED were chosen.

Another significant change is the inclusion of an introductory/background section. In the interim report this was placed in the general introduction section, however, it was subsequently decided that in order to provide more information and detail it should be moved to the literature review.

Wireless Sensor Networks: An Overview

According to Dulman *et al* a wireless sensor network is “basically, any collection of devices equipped with a processor, having sensing and communication capabilities and being able to organize themselves into a network created in an ad hoc manner” (Dulman *et al*, 2006).

Regarding the devices, wireless sensors (or nodes) are small (often embedded) devices that usually contain the following components; antenna and RF transceiver, CPU, memory (often flash), sensing device, and power (normally batteries) (Stavrou, 2005). Other than the actual wires and circuits, this is literally that these devices contain in terms of hardware. As nodes are generally battery powered and deployed unattended for long periods of time the amount of power available to them is tightly constrained (Yick *et al*, 2008).

Similarly, because of their size and lack of battery power, their processor is typically not very powerful, and the amount of memory available is extremely constrained (Yick *et al*, 2008). This is in contrast to other mobile devices such as laptop PCs or 'smart phones' which have sophisticated hardware, long battery-life, and (in the case of laptops) can sometimes rival desktop PCs in terms of memory and processing power.

As the name suggests, the primary purpose of a wireless sensor device is to perform some sort of sensing task (i.e. sense the temperature), collect the data (possibly do some basic processing) and then transmit it to a base station (usually a PC or more powerful node) (Stavrou, 2005). Because of their small size and low cost, wireless sensor devices are typically deployed for tasks unsuitable for human monitoring, for instance monitoring an environment for a sustained period of time, or collecting data from a dangerous environment such as a volcano (Yick *et al*, 2008).

Networking and Communications

Due to their limited energy and susceptibility to damage, wireless sensor devices are often deployed in large numbers over as wide a range as possible (Yick *et al*, 2008). This often leads to most nodes being unable to contact the base station directly due to limited energy and weak radio (Akyilidz *et al*, 2002).

One means of getting round this problem is to use multi-hop routing, this is when all nodes in the network act as both end users and routers in order to pool their resources and allow nodes to reach a target (i.e. the base station) they wouldn't be able to reach directly (Akyilidiz *et al*, 2002). This means that when a node is unable to reach it's target directly it will send its message to another node who will then pass it on to another node until it eventually reaches its destination (Akyilidiz *et al*, 2002).

Although wireless sensor networks are somewhat distinct from conventional wired networks, this does not mean that they do not have any characteristics in common at all. Although Quality of Service was never originally considered important for WSN's, as the devices have become increasingly advanced and continue to find new applications, it has started to become an important issues that has seen a significant amount of research dedicated to it.

The following section takes a look at wireless sensor QoS, first by providing a broad overview and history of QoS in general, and then taking a look at what makes WSN QoS unique.

An Introduction to WSN QoS

The previous section provided a broad overview of wireless sensor networks, their use, their technology, how they communicate with each other, and what makes them a unique and interesting technology. The last sub-section concerned itself with the communications aspects of WSNs and a very brief overview of issues ranging from media-access to routing and from energy efficiency to localisation.

Obviously this project concerns itself heavily with Quality of Service (or QoS), therefore it is important to acknowledge that this will be the focus of the report, however, that does not mean that QoS is the only aspect of importance. In fact, quite the opposite is true. As has been repeatedly emphasised, energy consumption is almost unequivocally the single most important factor for any practical sensor network, bluntly put; without any energy left there is no wireless sensor network.

However, does this mean that QoS must always be subservient to energy issues? Does it mean that because an approach uses more energy that another it should be immediately discarded? Is it perhaps possible to establish a means of determining desirable trade-off's between energy consumption and other factors? And perhaps most importantly, why are energy consumption and QoS supposedly seen as mutually exclusive? Can the two areas perhaps be combined?

The answer to the second is, yes, of course they can. Throughout this section and the next (Defining WSN QoS) numerous research papers that do just that will be surveyed and evaluated. However, before this can occur it is important to begin by introducing the general concept, and give a brief history of QoS.

A Brief History of QoS

One of the questions mentioned in the introduction was; why are energy consumption and QoS supposedly seen as mutually exclusive? The answer to this question perhaps lies in the origins, of QoS. Despite its status as a hot topic for research, QoS is nothing new and can traces its history back to before the internet revolution of the 90's, although until the mid 90's it was generally though of purely in terms of committed and excess bit rates (Pepelnjak, 2008).

In June 1994 a group of engineers published a Request For Comments (RFC) paper that would revolutionise the way in which QoS was perceived; Integrated Services (Pepelnjak, 2008). The original RFC (RFC 1633) defined integrated services (or IntServ as it is commonly known) as "a proposed extension to the Internet architecture and protocols to provide integrated services" with particular emphasis on meeting the needs emerging real-time services (i.e. streaming audio or teleconferencing) without negatively effecting non-real-time traffic (Pepelnjak, 2008).

Adiseshu et al (1998) describe IntServ as consisting of the following aspects; service classes, service-level indication, admission control, classification, and scheduling. Despite the significance, and of IntServ and its signalling protocol (RSVP) Adiseshu *et al* (1998) claim that due to issues relating to complexity, scalability, and lack of obvious billing model, IntServ has not been commercially successful.

Since its inception, a great deal of research has focussed on attempting to overcome these failures either through adaptation of IntServ, or development of a completely new model (Adiseshu *et al*, 2008). One of the new models that was directly influenced by IntServ is that of Differentiated Services (or DiffServ), which was introduced in December 1998 in RFC 2475 (Adiseshu *et al*, 1998). The primary difference between DiffServ and IntServ is that where IntServ relied on individual applications to assess and indicate their desired level of service, DiffServ simplifies this process greatly by making use of classes; groups of applications that require similar levels of service (Striegel and Manimaran, 2001).

Since its inception, DiffServ has undergone a number of changes, one of the most significant of which has been the increase in the number of classes that can be represented as a result of RFC

2474 which defined the Differentiated Services field that replaced the earlier type-of-services field.

It is important to emphasise, that while both IntServ and DiffServ provide means of marking packets (either directly by the application or based on classes) there is more to QoS than simply specifying service levels and marking packets. In order for QoS to work in a practical sense there needs to be some means of actually implementing these high-level policies on actual devices, and so this section will conclude with a brief examination of some traditional QoS mechanisms before going onto examine ones more appropriate for Wireless Sensor Networks.

Although conventional QoS is often considered to be synonymous with queuing strategies it also covers areas such as traffic shaping and policing (Cisco, 2003). The basic idea behind a queuing strategy is fairly simple and corresponds to a device's physical buffer with the simplest queuing strategy First In First Out (FIFO) ensuring packets that arrived first, leave first (Cisco, 2003). However, this is generally deemed too simplistic and so more complex strategies in which a single physical buffer/queue is split into a series of virtual sub-queues which get serviced in a different manner.

The level of service each receives is generally based on some form of marking – whether user-defined or standardised such as DiffServ – this ensures that packets deemed to be of greater priority (i.e. delay-sensitive traffic such as IP telephony) are treated better than packets with lower priority (Cisco, 2003). As queuing obviously increases delay strategies have been designed to avoid (or reduce) the need for queuing, while queuing strategies are generally known as congestion management another area of QoS seeks to prevent congestion, this is known as congestion avoidance (Kartvelishvili, 2003).

Generally speaking, congestion avoidance mechanisms can be broken down into two broad categories; traffic shaping and policing (Kartvelishvili, 2003). One of the most simple and influential shaping policies in which traffic enters the buffer at a variable rate and is shaped into a set rate (comparable to bucket with a hole in the bottom – it can't get filled at different rates but the water will leak out of it at the same rate) with excess packets simply being dropped (i.e. with the leaky bucket, when it is full water will flow over the top) (Wu and Chen, 1996).

As with the FIFO queuing mechanism, this was seen as being too simplistic and inefficient and so other schemes have been proposed, such as token bucket, a direct variation of leaky bucket which rewards devices that do not send bursts of traffic by issuing them with tokens, so that when a device wishes to send a packet it must 'pay' at a set rate (i.e. one token per packet sent) (Wu and Chen, 1996).

In contrast to traffic shaping policies which seek to smooth out bursts of traffic, traffic policing does not feature in any form of buffering and instead immediately drops packets that exceed the stated rate (Cisco, 2003). An example of a traffic policing tool is Committed Access Rate, which allows the users to set policies so that a certain action occurs if a packet conforms (usually it is sent (either to another device or to a higher layer) or passed onto the next level of the QoS strategy) or exceeds the user-defined rates (in the case of packets exceeding the rate they are almost always dropped (Kartvelishvili, 2003).

In order to meet the demands of bandwidth hungry applications such as teleconferencing, and multimedia streaming modern QoS strategies tend to include combination of all of the above elements, most frequently based on DiffServ markings (Cisco, 2003). In addition to trying to meet the demands of real-time networking, a new need area of QoS has arisen in rescent years; spurred on by the increase in mobile computational device, and the advances in wireless networking, ad-hoc networks have became an area of active interest, and as such researchers have begun looking at ways of implementing QoS technologies in this area.

QoS in Ad-Hoc Networks

Earlier on, it was stated that wireless sensor networks area form of ad-hoc networks, networks which are formed without infrastructure; for a specific purpose (from the Latin meaning of ad-hoc: for the purpose); routing often of a multi-hop nature (in which devices act as both routers an end-users); and usually (but not always (as is often the case with WSN's)) containing mobile devices. As such ad-hoc networks tend to function in a somewhat different manner from conventional wired and wireless networks (i.e. the need for all devices to act as both routers and end-users, or the use of dynamic routing) and therefore are subject to certain constraints that conventional networks aren't.

As ad-hoc networks are becoming more popular and prevalent, and as the range of applications which they are being used for increases, many have sought to examine ways in which QoS can

be implemented and managed in ad-hoc networks (Mohaparta *et al*, 2003). Like conventional QoS techniques, ad-hoc QoS can be approached at various levels corresponding to layers of the networking stack, for instance at the physical layers, Media Access Control (MAC) schemes such as IEEE 802.11's Distributed Coordination Function (DCF) (Mohapatra *et al*, 2003) or its successor 802.11e which actively focuses on WLAN and ad-hoc QoS (Xue and Ganz, 2003).

At higher-levels, a significant area of work focuses on implementing QoS in ad-hoc routing protocols (Mohapatra apart *et al*, 2003. Most ad-hoc routing protocols can be broken into three categories; reactive (try to find a route when needed), table-driven (store information about previosly discovered routes), and hybrid – a combination of the two Xue and Ganz, 2003). Of the approaches looked at by Mohapatra *et al* (2003), the first, CEDAR (Core Extraction Distributed Ad Hoc Routing protocol) appears to be a hybrid, but tends slightly towards the reactive category, as it attempts to dynamically calculate a centre-point for the network and exchange information regularly, but calculates route desirability (and thus determines which path it will take) only when it needs to send a packet.

The other approaches surveyed are not specific protocols, but are in fact categories of protocols, such as "Integrating QoS in Flooding-based Rout Discovery" (Moshapart *et al*, 2003) in which protocols use different variants of the classic flooding mechanism, usually involving some form of metric to determine what action to take when they receive a flooded message with regards to QoS (Mohapatra *et al*, 2003). The other two categories mentioned are "QoS Support using Bandwidth Calculations" (Mohapatra *et al*, 2003) in which protocols attempt to determine end-to-end bandwidth of links in order to make a choice, and "Multi-path QoS Routing" (Mohapatra et al, 2003) which is in contrast to standard ad-hoc routing as it attempts to find multiple paths to a source, in order to satisfy the desired level of QoS (Mohapatra *et al*, 2003).

In contrast to the methods surveyed by Mohapatra *et al* (2003), Xue and Ganz (2003) present an algorithm that is closely linked to the conventional QoS areas of resource-reservation and signalling. Their approach is based primarily on the reactive (on-demand) approach and attempts to reduce the amount of end-to-end signalling (a prominent feature in conventional QoS) in order to cope with limited bandwidth (Xu and Ganz, 2003). Their algorithm also features other interesting aspects such as violation detection and adjustment in which traffic that violates agreed parameters is penalised by either reducing or ending connections entirely (Xue and Ganz, 2003).

Challenges Facing WSN QoS

Although wireless sensor networks, like mesh networks and MANETs are classed as a type, or subset, of ad-hoc networks, just as traditional QoS mechanisms are not always directly applicable to ad-hoc networks, it is also possible that adaptations made for ad-hoc networks are not always suitable for wireless sensor networks.

The reason for this is the same as the reason why conventional QoS mechanisms didn't always fit ad-hoc networks; they contain different hardware and have different communication patterns. One notable difference between a wireless sensor network and the traditional ad-hoc network is mobility, in ad-hoc networks it is expected that devices (typically laptops or "smart" phones) will move frequently and unpredictably, although some WSNs feature mobility (i.e. those deployed in Un-manned vehicles) these are the exception rather than the rule (Akyilidiz *et al*, 2002).

The purpose of this sub-section is to explore the various challenges that wireless sensor networks present to QoS. In order to appreciate these challenges, a wide range of literature has been surveyed, and from this several broad categories have been established. Each of these categories contains background and a discussion of how this particular issue can affect WSN QoS. Some of these may ostensibly seem similar to ad-hoc and even traditional network constraints (i.e. bandwidth limitations), however this is expected, as the purpose is to examine all of the possible issues and constraints that affect WSN QoS.

Energy Consumption:

The reason that this has been listed first, is because it is undoubtedly the biggest constraint facing wireless sensor networks, as well as one of the most popular areas of WSN research (Yick *et al*, 2008). As most wireless sensor devices draw their power from a battery (which is of fixed capacity), which therefore limits how much energy they can consume (Slijepcevic and Potkonjak, 2004).

Min *et al* (2001) suggest that in addition to the available power being limited, another reason for the need to consume low amounts of energy is that unlike other small electronic devices (i.e. mobile phones) sensors are expected to fulfil a variety of roles (i.e. router, sensor, cluster-head)

while being deployed for a sustained period of time. Similarly, as nodes are often deployed 'in the wild' there exists the potential for nodes to be exposed to damage, or even suffer signal degradation to the extent other nodes believe the node is dead (Min *et al*, 2001).

As energy-efficiency is often measured in terms of node/network lifetime (how long a particular node or network is active for, of which there are various definitions) it is important to take into account that issues other than obvious energy/power issues could negatively effect the lifetime of a device or network.

Environment:

It has been mentioned on several occasions that wireless networks are frequently used for areas such as target tracking and environment monitoring, and as a result they are often deployed outside in potentially inimical conditions that can lead to their lifetime being shortened (Perillo Heinzelman, 2005). Certain deployment areas post a greater threat to network/device lifetime than others, i.e. those in which devices are exposed to extreme heat or pressure (Perillo and Heinzelman, 2005). For instance, Peterson *et al* (2009) have designed a middleware that is intended for use in a sensor network monitoring volcanic activity at Mt. St Helens.

Similarly, recent research has focussed on their potential for deployment in military applications, in which the risk of damage is significantly increased (Akyilidiz *et al*, 2002), such as the framework for utilising WSNs for naval applications explored by Enel and Martinet (2005).

Heterogeneity:

Heterogeneity can be defined in terms of platform and application heterogeneity. In terms of platform heterogeneity, Yick *et al* (2008) states that it arises from a lack of established standardisations, as current attempts at standardisation such as ZigBee are still relatively new. Middleware can be seen as a potential method of addressing the problems caused by platform heterogeneity by providing a common, platform-independent, framework that sits between the high-level applications and the network itself (Gowrishankar *et al*, 2002).

Application heterogeneity can be seen as arising from platform heterogeneity, Chen and Varshney (2004) explain that some applications might require more than one type of data to be

collected. For instance, an sensor network used for environment monitoring could contain nodes that sense the temperature, while other nodes sense the level of rain, in this case there are clearly two different types of data being collected (temperature and rainfall). Chen and Varshney (2004) further explain that in such sensor networks, in addition to more than one type of data being collected, the rate at which different types of data are collected could also vary.

Transmission Media:

In order for nodes to communicate wirelessly with each other and the base node all nodes are equipped with some form of transceiver (Akyilidiz *et al*, 2002). Unfortunately, this device tends to be responsible for the most power consumption, due to its need to regularly send, receive, and monitor channels (Zakaria, 2007).

In addition to issues relating to power consumption, another common drawback of wireless sensor radios is their limited range (Younis *et al*, 2004). This means that because of the size (usually in the range of 100s or 1000s) nodes are often unable to directly reach further away nodes, or more importantly, the base station (Younis *et al*, 2004). As such, communications in wireless sensor networks tend to be multi-hop (i.e. send a packet via intermediate devices) rather than end-to-end (send the packet directly to the source) (Younis *et al*, 2004). This is why it is often stated that wireless sensor nodes generally act as both routers and end-users.

Another major issue concerning radio transmission is radio irregularity (Zhou *et al*, 2004). Radio irregularity can affect WSN's in numerous ways ranging from networking and data-link layers (i.e. packet collisions, low reliability, etc.) to applications that require localisation and coverage estimation, due to the need of these applications to accurately analyse radio strength and coverage (Zhou *et al*, 2004). Radio irregularity is caused by several factors such as non-isotropic path loss (i.e. signal damage as a result of the signal being reflected, diffracted and scattered) and problems arising from different nodes using different transmitters (or even the same transmitters, but with different configurations) which causes nodes to transmit at different strengths/frequencies (Zhou *et al*, 2004).

Other Hardware Constraints:

This is in fact, something of a broad category that could technically contain the first category as well, because energy consumption is related to battery capacity, which is technically a hardware

issue. However, for this purpose hardware is taken to cover devices such as memory, processor, sensing apparatus, and transceiver (covered by the last constraint) which are the basic hardware components of most wireless sensor devices (Akyilidiz *et al*, 2002).

The needs of wireless sensor hardware will vary depending on their application (for instance, a target tracking application might require device mobility), but Akyildiz *et al* (2002) have identified the following as common requirements for wireless sensor devices; "consume extremely low power, operate in high volumetric densities, have low production cost and be dispensable, be autonomous and operate unattended, and be adaptive to the environment" (Akyilidiz *et al*, 2002).

Most of these requirements are not just specific and are linked to the general aims of sensor networking; autonomy, low power consumption, low cost (so as to allow for the deployment of large numbers, of which several are expected to be damaged), and the ability to record environmental data. It is important to observe, therefore, that many of these can be considered as requirements for not only the hardware itself, but also the software that uses it.

This is a logical assumption, as although a sensing device could be designed that in itself met all of these conditions, if the software that used it then failed to meet these conditions (i.e. used too much power, required an amount of memory that led to the node crashing, etc.) the network would not function optimally (if at all).

Looking back at the issues listed so far, it is worth noting that all of these could probably be placed in the category of causes of WSN issues (i.e. energy constraints can lead to node), however what has only been mentioned in passing is the actual effects that these causes have. Although some of these effects might be obvious (i.e. limited processing power and memory limits the complexity of what applications can be run), while some causes are responsible for multiple effects, or multiple causes are responsible for the same effect.

The next two issues are therefore both intended as effects of the previously discussed causes, they are not intended to map directly to any particular causes, and therefore are listed in an entirely arbitrary manner, like the causes (with the exception of energy consumption).

Dynamic Topology:

In addition to loss of energy, Chen and Varshney list “node failures, wireless link failures, node mobility, and node state transitions” as factors affecting the stability of sensor network's topology (Chen and Varshney, 2004). Akyildiz *et al* (2002) points out that in typical wireless sensor network applications thousands of nodes are deployed throughout the area, and once deployed are usually left unattended. Gowrishankar *et al* (2008) points out that sensor nodes becoming unavailable can affect the verisimilitude of readings gathered as can problems with the sink/base node which is usually responsible for providing some degree of coordination and control.

As wireless sensor networks are prone to a variety of failures ranging in severity from node death to temporary loss of communications, it is clear that the potential for a wireless sensor network's topology to vary in a dynamic and unpredictable fashion is an important issue that needs to be addressed.

Unconventional Routing:

This was touched upon earlier when multi-hop routing was discussed as a means of getting round limited range of transmission media of most nodes. To recap, multi-hop routing involves nodes sending their messages indirectly via intermediate nodes, rather than directly to the source. Stankovic (2006) states that a direct result of this is that most Internet and MANET routing protocols are not directly applicable to wireless sensor networks. Stankovic (2006) asserts that Internet routing does not work due to the unreliability of WSN links, while Yick *et al* (2008) adds that sensor networks do not normally use IP addresses.

Due to wireless sensor networks being thought of as a subset of ad-hoc networks and sharing a number of similarities and constraints, it is perhaps more surprising that very few ad-hoc routing protocols are suitable for use with sensor networks. Stankovic (2006) argues that the main reason for this is due to ad-hoc routing protocol's need for symmetric links, whereas in sensor networks are often asymmetric and liable to frequent change.

As with almost all aspects of WSN's, energy consumption is a major factor in routing. Although it is argued that multi-hop routing (in addition to improving reliability) also conserves energy,

due to the reduced transmission range needed to send a packet, it could also be argued that intermediate nodes are required to do more work than they would normally Younis *et al*, 2004). Despite the increased complexity resulting from all nodes being potential routers, Younis *et al* (2004) argue that unless all nodes are placed near the base station, overall, multi-hop routing is more effective than direct transmission.

Within multi-hop routing systems, a number of protocols such as the work of Schugers and Srivastava (2001) focus on directly using routing as a means of conserving energy, by basing routing decisions on factors such as transmission cost (i.e. will path x require more energy than path b) as well as residual energy of remaining nodes (Yick *et al*, 2008).

In addition to multi-hop and other constraints related to energy consumption, another reason that conventional or ad-hoc routing systems are inappropriate for WSN's is as a result of device (and sometimes application) heterogeneity which was discussed earlier. As a result of nodes having varying resources, a number of approaches have been proposed that treat nodes differently based on their resources. An example of such an approach is clustering (which will be described in detail later) in which the entire network is split into subsets (clusters) of nodes which are controlled by a cluster-head elected by the network based on factors such as processing power, signal range, or remaining energy (Dai and Liu, 2009).

Existing Research

A large amount of research in wireless sensor network has involved routing protocols that are able to provide, or are at least are aware of, application QoS such as the approach previously mentioned of Lu *et al* (2007) which uses cost functions to select the best routes in terms of meeting QoS standards as well as conserving energy.

Another QoS-aware routing algorithm is proposed by Gelenbe and Ngai (2008) who have developed a randomised routing algorithm in order to provide priority to important traffic during periods of network congestion.

A practically-oriented approach is that of Howitt *et al* (2006) who propose a solution for Wireless Industrial Sensor Networks that is based on engineering control theory and involves

using wireless sensors to monitor industrial applications, the example in the paper being that of a factory.

At a higher-level, a number of research papers focussing on developing a middleware that sits between wireless sensor applications and the network itself. An example of one is described by de Freitas *et al* (de Freitas *et al*, 2008) that attempts to address the heterogeneity of the network which can potentially contain a number of low-end sensors alongside more sophisticated equipment by proposing an alternative to resource-intensive CORBA by providing a highly-adaptable middleware that can run on nodes of varying capabilities.

A similar concept is presented by Troubleyn (Troubleyn *et al*, 2008); a middleware for wireless sensor networks that uses modularity to cope with node heterogeneity by making certain modules optional, meaning that only nodes with significant processing power and energy will run them.

Heterogeneity is also addressed by Yaghmaee and Adjeroh (Yaghmaee and Adjeroh, 2008) who base their approach on taking an existing technology (DiffServ) an adapting it for wireless sensor networks in an attempt to providing different levels of service for different types of traffic.

Several researchers have stressed the importance of a self-adapting network that does not rely on humans for configuration, for instance, an adaptive mobile agent in conjunction with an attribute-based addressing scheme is proposed as an effective means of meeting QoS demands (Spadoni *et al*, 2009). Similarly, Barbancho *et al* (Barbancho *et al*, 2006) present an algorithm called SIR: Sensor Intelligence Routing, which uses neural networks located in all nodes in order to optimise the decision-making element of the routing process.

A significant portion of work (including some of the research mentioned above) focuses on developing methods of calculation for node configurations, link metrics, etc. while some of these develop their own such calculations, several researchers have adapted existing mathematical and logical approaches for use in wireless sensor network QoS.

Hoes *et al* (Hoes *et al*, 2009) use Pareto algebra to efficiently calculate node configurations in order to meet application-level QoS demands. Fuzzy-logic is used in order to estimate network

calculation and then reconfigure nodes in order to reduce congestion (Munir *et al*, 2007). Fuzzy-logic is also used by Xia *et al* (Xia *et al*, 2007) who employ it in Wireless Sensor Actuator Networks to adjust the sampling rate of sensor nodes in order to ensure more packets are received by their deadline. Hamdy and El-Madbouly (Hamdy and El-Madbouly, 2007) base their approach on the work of Xia *et al* (Xia *et al*, 2007), by combing the fuzzy-logic approach with a genetic algorithm.

Clustering

Due to several constraints that face wireless sensor networks, a great deal of protocols make some use of multi-hop communications, in contrast to the single-hop communications used by conventional networks. Although it is often recognised as a necessary compromise, some researchers such as Guo *et al* (2006) argue that multi-hopping significantly increases the amount of message forwarding carried out by intermediate nodes and as such is a drain on resources.

As an alternative to the conventional multi-hop approach taken to WSN communications they propose a hierarchical clustering algorithm in which nodes form group (known as clusters) and send their data to a cluster-head who is responsible for sending the data on to the base node (Guo et al, 2006). However, the approach that is used by Guo et al (2006) is actually quite different from the 'typical' clustering strategies, as it makes use of dynamic clustering (in response to a particular event).

Akyilidiz et al (2002) describe the typical characteristics of a clustering protocol as being the grouping of a network into small sub-groups (clusters), which are lead by a cluster-head who is responsible for 'managing' the group and performing functions such as data aggregation and energy management (i.e. allocating tasks to members based on their remaining energy).

Although it was stated that the protocol proposed by Guo et al (2006) was atypical due to its use of dynamic clustering, it is only appropriate that an explanation of how cluster formation works in 'typical' clustering protocols as well as presenting other alternatives in addition to the work of Guo et al (2006).

Cluster Formation

An important point to consider when comparing clustering schemes is how the clusters are formed. This is one of the most important differences between schemes and is usually indicative of the overall aims of the scheme (i.e. energy conservation, communication reliability, etc.). Dai and Liu (2009) state that the two most common methods of cluster formation are nodes electing themselves as cluster-heads and other nodes then joining, or for the entire network to be partitioned into smaller areas and within each of these areas a node is chosen as that area's cluster-head.

Within the first category there are still major differences in cluster formation between clustering algorithms. Chaman and Pierre (2009) present an algorithm called 'Energy-Efficient Cluster Formation protocol (EECF)', in which cluster-head suitability is assesed based on the triple-constraint of current energy-level, node degree, and distance from the base node. To summarise, the clustering process begins with nodes dividing their neighbours based on whether their score is higher or lower than their own and when this is complete they will either join a cluster led by a node with a higher score than theirs, or become a cluster-head themselves if they are elected by another node (Chaman and Pierre, 2009).

Although this is only a summary of the entire process that leaves out a lot of the complexity, it could be said to be fairly typical of the steps taken by most election-oriented clustering algorithms in that nodes attempt to gain an overview of their neighbours capabilities by exchanging information about themselves with others nodes. The information they receive from other nodes is used as their basis for electing a cluster-head. Once all cluster-heads have been elected, nodes will again use the information they have learned, this time to choose which cluster-head to join. The exception being if a node is itself elected a cluster.

The work of Deng et al (2008) introduces the concept of nodes being randomly appointed as cluster initiators who are responsible for sending messages to other nodes in the network to inform them of the cluster's existence. Nodes are either permitted or denied entry to the cluster based on the Hausdorff distance between the prospective member node and the cluster and if they are denied entry they become cluster initiators and being transmitting messages to other nodes (Deng et al 2008).

Another variation to the standard model of cluster formation that was discussed earlier is that of Guo et al (2006), whose approach involves the introduction of an artificial neural network in order to allow nodes to react to a significant event and transmit data to the base station (Guo et al, 2006). In the proposed algorithm, when a significant event occurs nodes will transmit information about it one-hop to their neighbours, but when they receive a certain amount of messages from their neighbours they will then become the cluster-head to whom all other nodes will forward their messages to (Guo et al, 2006).

Guo et al (2006) argue that the traditional model of WSN clustering where cluster-heads are decided before sensing commences runs contrary to the goal of data-centric communications as it does not take into account "data distribution and the similarity of the data" (Guo et al, 2006).

Another, alternative to traditional clustering measures is that proposed by Dai and Liu (2009) which radically diverges from the two standard models of clustering described above. Instead they propose a solution in which an algorithm is used to determine which nodes will act as cluster-heads in order to ensure maximum coverage and energy distribution (Dai and Liu, 2009).

So far the actual concept of cluster-head election has only briefly been touched on. Although it was mentioned that in approaches such as that used by Chaman and Pierre (2009) cluster-heads are elected based on an exchange of data about themselves, which then forms the basis of a nodes decision on whether or not to elect another node as cluster-head.

An obvious problem with this approach is that it can lead to a large volume of traffic as newly initialised nodes attempt to share information about themselves with other nodes. In response to this, the approach used by (Estrin et al, 1999) attempts to limit the number of advertisement packets by determining a radius in which sensors transmit advertisements. This radius is based on a hop-count, and in addition to reducing the number of messages being sent, it also ensures that clustering is local, as there is no point in having nodes at opposite ends of the network trying to form a cluster (Estrin et al, 1999).

With regards to what basis cluster-heads are elected on, this largely varies from protocol to protocol, although popular metrics include latency, communication cost, path cost, and energy levels (Abbasi and Younis, 2007). An interesting variation to the election processes described so

far is that of LEACH (Low-Energy Adaptive Clustering Hierarchy) which puts a great deal of emphasis on energy conservation so that sensors nominate themselves rather than other nodes as cluster-heads while non-cluster-head nodes then make a decision on which cluster to join based on how much energy it takes to reach that cluster-head compared to others (Heinzelman et al, 2000).

With the cluster now formed and all nodes either a member of a cluster or a cluster-head themselves, what happens next? What is its use and purpose of the cluster-head? These two questions are related to the same answer and that answer is the functionality of the cluster-head. Taleghan et al (2007) that cluster-heads are responsible for collecting data from cluster members. Taleghan et al (2007) also state that the cluster-head is then responsible for taking some sort of action based on these results whereas the model Choi et al (2009) has cluster-heads forwarding data collected from member nodes onto a base node.

Although there is no definitive answer as to which approach is more prevalent, the majority of the relevant literature reviewed for this project uses a similar definition of the basic functionalities of a cluster-head as Choi et al (2009) use. For instance, the description of cluster-head functionality offered by Choe et al (2009) takes the basic description of Cho et al (2009) and expands on it by adding that in addition to collecting and forwarding data from member nodes, it states that cluster-head "manages the data exchanges between sensor nodes" (Choe et al, 2009).

This description initially appears rather vague, however, it is later clarified as meaning the responsibility of the cluster-head has been extended to give cluster-heads the responsibility of determining when member nodes will collect and send data (Choe et al, 2009). Cluster-heads often have the responsibility of scheduling or coordinating node activity so as to decide when to receive data from member nodes, or when to send it to the base station once stored. The next, somewhat broad, sub-section covers everything from sensor scheduling and selection, to routing of received data.

Cluster Management

In the system developed by Biyu et al (2005) the cluster-head takes on a role similar to that of its counterpart in Choe et al (2009), being responsible for determining a control value which member nodes will use to alter their sensing rates. This approach requires that the cluster-head

not only calculate and transmit such a value to its member nodes, but that it does so based on the QoS-requirements of the network and is also responsible for determining which nodes are alive and which are dead (Biyu et al, 2005). While both systems might appear conceptually similar as the cluster-head plays a similar role in which it is responsible for scheduling members transmissions, there are several significant differences between the schemes used by Choe et al (2009) and Biyu et al (2005).

One of the most fundamental differences between the two approaches is the degree of responsibility that is placed on cluster-heads. While Choe et al (2009) favour giving cluster-heads complete responsibility for determining what nodes can report data and when, the method of scheduling used by Biyu et al (2005) is more decentralised as cluster-heads determine a control value which cluster members then use to calculate when they will next send data to the cluster-head. This difference is perhaps indicative of the overall methodologies of the two protocols; whereas the former places more emphasis on the cluster-head managing the cluster (Choe et al, 2009) the later grants cluster members a greater degree of independence and autonomy (Biyu et al, 2005.

This is evident in the implementation of the scheduling; while cluster-heads receive and process requests from their members in order to decide whether or not to allocate a particular time slot to the requesting node in the work of Choe et al (2009) Biyu et al (2005) use a more democratic scheme. In this scheme the cluster-head determines a control value for its cluster and transmits this to all active nodes by sending it via acknowledgement messages, upon receiving this the nodes then use this to determine when they will next report to the cluster-head, therefore giving the individual nodes a greater element of control over their report scheduling (Biyu et al, 2005).

Other differences between the two protocols are mainly related to exactly how the scheduling is calculated as both take QoS into account, but while the calculations used by Choe et al (2009) are relatively simple, Biyu et al (2005) favour a more complex mathematical model derived from the Bernoulli benchmark approach. While it could be said that the approach taken by Biyu et al (2005) is more complex due to its schedule calculation techniques and distribution of responsibility between cluster-heads and members, it is important to note that the solution proposed by Choe et al (2009) is more complex.

In fact, it could be said that this approach is closer to a complete cluster management solution, as cluster-heads are not only responsible scheduling of cluster member reporting, but also actively

participate in the routing process (Choe et al, 2009) whereas the work of Biyu et al (2005) focusses on scheduling and completely ignores routing. As with scheduling, Choe et al (2009) base their routing strategy on QoS (their solution is called QoS-aware data reporting tree construction scheme (QRT)) in which cluster-heads determine the shortest path to the base node and then form a spanning tree by exchanging route information with other cluster-heads. The spanning-tree algorithm has been modified so as to take the QoS metrics of delay and traffic load into account, thus allowing cluster-heads to select paths with QoS in mind (Choe et al, 2009).

From the perspective of cluster-heads, a simpler approach to routing is taken by Manjeshwar and Agrawal (2001) who present Threshold Energy Efficient Network protocol (TEEN) which is an early attempt at creating a reactive routing protocol for wireless sensor networks. Unlike, the proactive, tree-building approach favoured by Choe et al (2009) TEEN places more emphasis on trying to save energy by reducing the number of messages sent by nodes than on implementing a complicated routing scheme, as it favours a reactive approach in which cluster-heads find a route to the base node as soon as they receive packets from their cluster members (Manjeshwar and Agrawal, 2001).

So far, most of the forms of cluster management that have been looked at so far have involved some degree of scheduling and routing. Although scheduling features in a large number of clustering protocols, it is not the only form of management that a cluster-head can perform. It was mentioned earlier on that a common function of clustering is the use of data aggregation in which ordinary nodes send their data to the cluster-head who combines the readings from its members and sends the combined readings to the base station (Akyilidiz et al, 2002). In addition to being significant in its own right, alongside clustering, aggregation is also one of the areas being investigated in this project, therefore rather than simply fit it in this section with the other functions of cluster-heads, the next section will take a detailed look at various approaches to data aggregation.

Data Aggregation

Data aggregation is an important feature in most of the clustering schemes surveyed so far, however, until this point it has only been briefly mentioned and no detail has been given. Krishnamachari et al (2002) state that data aggregation's popularity in WSN is research is due to its ability to “combine the data coming from different sources enroute” and as a result aggregation is useful for “eliminating redundancy, minimizing the number of transmissions and thus saving energy” (Krishnamachari et al, 2002).

As earlier sections of this report showed, energy conservation is always an important issue in WSN design due to the (typically) limited battery power of sensor devices (Yick et al, 2008), therefore any method that helps decrease energy consumption will always be viewed in a positive light. However, this brings up two important questions; how exactly does data aggregation work? And; does it have any negative effects?

Mhatre and Rosenborg (2004) state that the standard model for data aggregation operates on the assumption that the cluster-head receives sensed data from all of its member nodes and performs some form of processing to combine these separate data reports into one fixed size report. The advantages of using this basic, many-to-one method of data aggregation are the use of a standardised method and the ability of the application using the network to determine what level of aggregation is required (Mhatre and Rosenborg, 2004).

The data aggregation schemes examined by Krishnamachari et al (2002) all work on this assumption, however, there are some slight variations on this model, as some of the schemes classed as suboptimal do not use cluster-head aggregation, such as the Center at Nearest Source method in which all aggregation takes place at the node nearest the base (Krishnamachari et al ,2002).

Cam et al (2006) also present another comprehensive framework in which data aggregation is combined with scheduling to ensure that where multiple nodes are transmitting the same data only one of these nodes need to be active. This is accomplished by determining which nodes have overlapping sensor ranges and coordinating feedback accordingly (Cam et al, 20060. The actual method of aggregation is interesting as it is performed using pattern codes to perform advanced elimination of redundant data rather than simply dropping information that appears the same (Cam et al, 2006).

Despite the advantages of the standard aggregation principle, the use of a standardised aggregated packet size can also be seen as a disadvantage as it might not actually be possible to combine all of the received data into a packet of the required size, or doing so may seriously compromise the accuracy of the collected data (Mhatre and Rosenborg, 2004). The issue of accuracy versus packet size is the subject of much debate and research with numerous proposals trying to find a desirable trade-off between the two constraints.

The model proposed by Mhatre and Rosenborg (2004) proposes that the number of packets a cluster-head receives and the degree of aggregation that is permissible determine the number of packets required for a cluster-head to aggregate all data they receive. This method of data aggregation is not presented in isolation, but is instead part of a comprehensive framework that also covers cluster-formation and node communication (Mhatre and Rosenborg, 2004). Particular emphasis is given to how these areas affect each other, as it is stressed that the above method of determining packets required for data aggregation should be used to determine the number of cluster-heads so as to spread the strain amongst multiple nodes (Mhatre and Rosenborg, 2004).

The HEED Protocol

Earlier on in this literature review, the concepts of clustering were introduced and several related topics such as cluster formation and cluster management were examined, often using examples of existing protocols for illustration. While the purpose of the previous section was to introduce, examine, and justify the use of clustering in wireless sensor networks, the purpose of this section is to examine a specific clustering protocol. HEED forms part of the iHEED application that is itself the focus of the evaluation detailed in the Methodology section

HEED stands for 'Hybrid Energy-Efficient Distributed clustering' and is intended to provide a cluster-based approach for guaranteeing node connectivity in ad hoc wireless sensor networks without draining nodes of their energy (Younis and Fahmy, 2004). HEED is equally compatible with homogeneous and heterogeneous networks as the protocol is not designed for specific deployments or types of node (Younis and Fahmy, 2004).

HEED follows a model in which cluster lifetime is not infinite, but is instead determined by a factor known as 'total network operation' which varies depending on the application of the network (Younis and Fahmy, 2004). Regardless of whether clustering is taking place for the first or fifth time the process of cluster formation is exactly the same; all nodes calculate their probability of becoming a cluster-head and then share data with their neighbours in order to elect or be elected as a cluster-head (Younis and Fahmy, 2004). Throughout the clustering process, a node's probability of becoming a cluster-head will change as it progresses through the stages of the algorithm and possibly receives elections from other nodes (Younis and Fahmy, 2004).

Nodes that do not have a cluster-head as a neighbour automatically become cluster-heads if they complete the algorithm without finding a cluster-head (Younis and Fahmy, 2004) as this ensures that all nodes are either cluster-heads or cluster members, thus all nodes are in a cluster of some sort.

Except in the case of nodes that are unable to connect to an existing cluster-head, all cluster-heads are elected by their fellow nodes (Younis and Fahmy, 2004), but on what basis does this election take place? In their survey of WSN clustering protocol, Abbasi and M. Younis (2007) state that cluster-head election is based on node energy such that only nodes that have a high level of remaining energy are elected cluster-heads (Abbasi and Younis, 2007). More specifically, the primary metric for deciding cluster-head suitability is that of current energy levels, and once potential cluster-heads have been decided upon using this, a secondary metric relating to the cost of communications between clusters is used as a tiebreaker (Younis and Fahmy, 2004).

As HEED's cluster formation algorithm places emphasis on prolonging the overall network lifetime by distributing energy consumption, this leads to the potential for important sensors to spend more energy in order to extend the lifetime of less important sensors than they would in a scheme based on prioritisation (Soro and Heinzelman, 2009). However, it is important to emphasise, that HEED explicitly states that it makes no assumptions about device capabilities (Younis and Fahmy, 2004) therefore, by extension the developers have chosen to sacrifice the potential for preserving more powerful nodes in order to extend the overall lifetime of the entire network.

As mentioned in the introduction section of this report, although its authors state that HEED is suitable for working with and supporting data aggregation (Younis and Fahmy, 2004) it does not actually contain any data aggregation facilities. However, the iHEED application which is an implementation of the HEED clustering protocol in TinyOS does perform data aggregation (Younis and Fahmy, 2005), and an overview and examination of this application will form the basis of the next section.

The iHEED Application

In 2005, iHEED was designed and deployed on Mica2 and Mica2Dot nodes running the TinyOS operating system (Younis and Fahmy, 2005). iHEED is a TinyOS application the implements

HEED clustering, multi-hop routing, and data aggregation. As iHEED has been tested on real nodes rather than just simulated, it has been proved that it is able to cope with the hardware and other constraints facing such devices (Younis and Fahmy, 2005) and is therefore a viable WSN application.

So far it has been mentioned that iHEED is an application that will be simulated within the TinyOS environment, however although this is technically correct, it is important to emphasise that TinyOS is not an actual simulator, but is instead an operating system designed for wireless sensor devices (Levis et al, 2006). Although the original iHEED application was originally written for the TinyOS platform, this is not the only reason for its use, and the next section will provide a brief overview of the advantages and disadvantages of using the TinyOS platform.

TinyOS

One of the main reasons for choosing TinyOS is its popularity; TinyOS is "by far the most popular [wireless sensor] OS adapted by both researchers and industry alike" (Gowrishankar et al, 2008).

This is an important advantage as it means that this project will be analysing an application (iHEED) that has been built for one of the most widely used wireless sensor operating systems. Similarly, carrying out the simulation in TinyOS as well (via TOSSIM) allows simulation of not just the underlying protocol (HEED) but of the entire platform on which it will be deployed in research and industrial activities. Similarly, as stated in the introduction, to date there have been no published attempts at simulating iHEED, therefore, another important aspect of this project is modifying the iHEED application to run in TOSSIM simulator.

Why is TinyOS so popular, though? An obvious reasons for its popularity is that it is one of the few dedicated wireless sensor operating system, described by its development team as "a flexible, application-specific operating system for sensor networks (Levis et al, 2006). TinyOS was originally designed as an operating system for the Berkeley MICA series of wireless sensor devices with design emphasis being placed on concurrency and resource management (Gowrishankar et al, 2008).

This means that from its inception, TinyOS was designed with the hardware constraints of wireless sensor devices in mind. As opposed to standard embedded device operating systems, which would take a more general approach to hardware constraints. In order to cope with the hardware constraints facing sensor devices (as discussed earlier in this report) TinyOS's development team state that “many applications fit within 16KB of memory, and the core OS is 400 bytes” (Levis et al, 2006).

In order to allow for low memory and CPU usage, the architecture of TinyOS is defined as component-based (Levis et al, 2006). This means that developers only use components relevant to their application which stops nodes from having to lode up unnecessary operating system code, and thus helps reduce the amount of memory and processing required to run an application Gowrishankar et al (2008).

Regarding the programming aspect, both the operating system and applications are written using the nesC language which is a variant of C, designed to meet the specific needs and challenges of programming embedded sensor networks (Gay et al, 2003). As nesC has been designed with embedded devices such as wireless sensors in mind it not only provides efficiency, but also safety and reliability through measures such as eliminating dynamic memory allocation and pointers, data-race detection, and a safe concurrency framework (Gay et al, 2003).

Despite its numerous advantages, TinyOS is not perfect. When IBM were conducting research for their Secure Trade Lane project, they considered using a system based on wireless nodes running TinyOS (Thomsen and Husemann, 2004). However, this approach was abandoned due to TinyOS's incompatibility with the 802.15.4 FFD MAC layer, as well as its unsatisfactory execution times (Thomsen and Husemann, 2004).

Another issue with TinyOS is it's inability to perform real-time scheduling as a result of it only implementing an unsuitable First In First Out (FIFO) queue (Zhao et al, 2009). However, Zhao et al (2009) successfully modify TinyOS to implement a soft, real-time scheduler that is an improvement on the standard scheduling mechanism. Similarly, the research of Peterson et al (2009) focusses on implementing a variation of the Weighted Fair Queue algorithm (titled Dynamic-Weighted Fair Queue) and testing proved their approach was viable.

As the work of this project does not require a particular MAC protocol to be used, nor is there any need for real-time scheduling, considering the iHEED application is a simple environment monitoring application in which nodes merely sense and send (Younis and Fahmy, 2005) these are not particularly significant disadvantages. On the other hand, by using the TinyOS platform and its TOSSIM simulator, this allows for the simulation of not just the iHEED application and its underlying protocols, but also the detailed simulation of nodes running the application.

Similarly, due to the safe, lightweight, architecture and its status as the most popular WSN operating system (Gowrishankar et al, 2008) TinyOS is a viable option for many commercial and academic WSN deployments, therefore by conducting this investigation on the TinyOS platform, the results (from both the simulation, and modification of the iHEED application) will have greater significance than if the work was conducted on a more obscure/experimental platform.

Surge

It has been mentioned several times that iHEED is based on the Surge application which comes with TinyOS 1.x, however until now Surge has only received a passing mentioned. As iHEED is derived from Surge it contains several characteristics and mechanisms that are inherent to Surge, therefore, to avoid unnecessary repetition, the Surge application will be discussed in detail in the next sub-section with the sub-section following that focussing on iHEED.

Like the iHEED, Surge is a data gathering application that uses multi-hop routing to relay data from nodes to the base station at a fixed rate (Malesci and Madden, 2006). More specifically, the multi-hop routing used by Surge is dynamic and tree-based, that is to say, that although nodes attempt to build the tree before sending packets (Sammapun *et al*, 2007) they will make their decision on which parent to send their readings to dynamically (Malesci and Madden, 2006) and attempt to find a new parent if the link quality falls below a designated threshold (Malesci and Madden, 2006).

Unlike conventional proactive or reactive routing protocols which both rely on explicit mechanisms for sharing data about routes between nodes, nodes running Surge monitor transmissions in order to select a parent (Sammapun *et al*, 2007). In order to ensure that nodes are using the most appropriate parent, nodes continually monitor the link quality and will attempt to find a better parent if the the state of the link falls below a certain level. The measurement of

link quality is achieved through the use of Active Message (AM) acknowledgements so that the fewer transmissions that go unacknowledged the higher the quality of the link.

In addition to being used as a means of grading link quality, AM acknowledgements are, perhaps more importantly, used as a more immediate means of achieving reliability; if a node receives no acknowledgement from the base station then they will attempt to retransmit the data (Malesci and Madden, 2006). This is contrary to the approach taken by most researchers who believe that retransmission is an unnecessary waste of energy and other resources (Malesci and Madden, 2006).

A more technically-oriented, lower level, description of the Surge application is provided Kothari *et al* (2008) and the following paragraphs is primarily based on their work as well as observations made from examining the Surge source code.

As stated by Malesci and Madden (2006) nodes running Surge periodically sense their environment and transmit this data to the base node at a fixed rate, the catalyst for this process is the Timer.fired event which occurs when the timer reaches 0 (Kothari *et al* 2008). When this event takes place (in addition to performing other functions such as restarting the timer, etc,) the ADC.getData task is called which tells the node to gather data and when this is completed the ADC.dataReady event occurs to tell Surge that the data has been collected (Kothari *et al* 2008).

Once the data is ready Surge then uses the Send.send function in order to begin the process of transmitting the data to the base station, first ensuring that the variable gfSendBusy is FALSE (i.e. the node is no trying to send another packet), setting it to TRUE (to prevent the node sending another packet until the sensed data is sent). Once the data has been correctly sent the Send.sendDone event is triggered and gfSendBusy is set to FALSE (Kothari *et al* 2008).

iHEED

It was mentioned in the previous sub-section, that the iHEED application bears a number of strong similarities to the original Surge application of which it is a modification, however there are several main differences between iHEED and Surge, the most notable of which are the addition of data aggregation and clustering (Younis and Fahmy, 2005). However these are not the only alterations that have been made, other features include a means of estimating energy

consumption, as well as more advanced sleep function that allows nodes deemed surplus to requirements to conserve energy (Younis and Fahmy, 2005).

As the clustering method used by iHEED is largely an implementation of the HEED protocol proposed by Younis and Fahmy (2004) it has been deemed unnecessary to provide a separate explanation of the conceptual aspects of it as these have already been discussed. However, although the clustering portion of iHEED has already been covered, very little has been said about the data aggregation iHEED uses, so before providing an overview of the technical aspects of the iHEED application, a brief overview of how iHEED's data aggregation works will be provided.

The form of data aggregation used by iHEED is based on the concept of an aggregation-tree in which nodes forward their data to cluster-heads who are responsible for aggregating the data and then sending it onto the base node (the root) (Younis and Fahmy, 2005). When they receive sensed data, cluster-heads perform aggregation by adding the sum of all data values together for average and sum operators, or if maximum and minimum are being used then the minimum value is added (Younis and Fahmy, 2005).

As well as adding data together, cluster-heads collect the number of different readings they have aggregated and place this in the aggregated transmission (Younis and Fahmy, 2005). This means that when the collected readings are received, the average can be calculated by dividing the data by the number of readings, or the sum and maximum/minimum can be derived from the actual data (Younis and Fahmy, 2005).

Although it has been said that iHEED is a modification of Surge, it is not just Surge that has been modified, but several system files relating to muli-hop routing operation have also been modified in order to allow for implementation of functions such as clustering and data aggregation (Younis and Fahmy, 2005). The 'About.txt' file that comes with the source code contains a list of all the new or modified files iHEED contains, and a copy of this can be found in Appendix 1.

Aside from the Surge application file, most of the modifications have been made to the multi-hop routing files which were briefly mentioned in the Surge overview, but in order to avoid unnecessary detail were not examined in detail. Again, the exact operation of this aspect is not

really relevant to this report, and in attempting to explain the operation of Surge, attempts will be made to keep detail as high-level as possible.

The four main aspects of iHEED are defined as; application, routing engine, and routing logic. (Younis and Fahmy, 2005). The main purpose of the routing engine is to take of forwarding packets, and intercepting sensor readings (if the node is a cluster-head) and sending them onto the application layer (Younis and Fahmy, 2005). When the packet reaches the application layer it is then aggregated according to aggregation configuration, and the number of separate readings aggregated into the packet is updated (Younis and Fahmy, 2005). The packet is then sent back down to the routing engine layer to be forwarded along the tree to the sink (Younis and Fahmy, 2005). In order to perform the actual forwarding of packets, the routing engine must consult the routing logic module in order to determine the next hop(Younis and Fahmy, 2005).

The routing logic module is primarily responsible for providing the actual routing algorithm used by nodes to construct a tree and consists of two sub-modules; clustering logic and parent selection (Younis and Fahmy, 2005). The first of these sub-modules contains the instructions necessary to implement HEED clustering, and once clustering is complete forms a tree between cluster-heads (Younis and Fahmy, 2005). The parent selection sub-module's main task is to perform link evaluation using data losses and link symmetry in order to help decide which node to send data to (Younis and Fahmy, 2005).

It is also the routing logic module that is responsible for initiating clustering, as it contains Timer2, which when it expires signals that a node is no longer a member of a cluster and enters the clustering process described in the HEED section (Younis and Fahmy, 2005). Once clustering is complete and the node has either became a cluster-head or joined another cluster it will force Timer1 (the timer responsible for sending route updates) to zero, thus notifying its neighbours of its status (Younis and Fahmy, 2005). The other timer, Timer3 is responsible for ensuring that stages of the clustering process do not exceed a set duration, and when Timer3 expires the clustering process will move to the next stage (Younis and Fahmy, 2005).

Defining QoS

The first section of this literature review provided a general background to wireless sensor networks, their hardware, their operation, and their use, while the second focused on QoS; first with a discussion of QoS in general, before moving towards WSN's with an examination of what

makes them different from other types of network and ending with a brief review of a wide range of research related to WSN QoS.

The purpose of this section is to build on the work of these two sections, by combining both the general knowledge of WNS's as well as how their needs translate into QoS requirements and then determine appropriate definitions of QoS. The reason that WSN QoS needs to be defined in this section is because, to date there has been no universally accepted definition of QoS. Although it could be argued that there are no universally accepted definitions of QoS for conventional networks, and that the term is somewhat open to interpretation, QoS for conventional networks is still more rigidly defined and standardised than it is for WSN's.

Traditional QoS Metrics

In order to better understand how QoS is perceived, this sub-section will look at popular views of traditional QoS metrics. Just as the last section contained a discussion of the history of QoS in order to give background, this section will be used in order to examine how QoS has been defined, before looking at more specific attempts to define QoS for WSN's.

As was discussed in earlier sections, the popularity of QoS has arisen from the need to have networks support real-time, delay-sensitive traffic in addition to traditional best-effort data. The result of this is that a great deal of traditional QoS metrics are related to the transmission of real-time traffic. This is logical, when an email is sent so long as it eventually reaches its destination the time it takes is not usually significant, whereas when a video is being streamed, if the connection is too slow and the stream arrives seconds too late the user will notice deterioration in quality.

One of the most significant, and also frequently used, metrics is delay. This is defined as a measurement of the time between the source sending and the destination receiving (Goldsmith et al , 2006). In a real-time application, a high degree of delay is undesirable as it means that packets are taking too long to reach the source, which could lead to a high level of buffering between packets (Goldsmith et al, 2006). Another popular metric and one that is directly related to delay is jitter, or delay variation (Goldsmith et al, 2006). Jitter is defined as the difference in delay between packets that compromise a flow (Goldsmith et al, 2006). That is to say, imagine there is a group of packets that make up streaming video (i.e. one packet per frame), if jitter

exists then these packets will suffer different levels of delay which could lead to them arriving out of sequence, and thus being played in the wrong order.

Of the main QoS metrics, two similar, but not identical metrics that examine the similar areas are packet loss and reliability (El-Gendy et al, 2003). The first of these, packet loss, is the number of packets that have failed to reach their destination, if retransmission is being used then this does not alter the number of failed transmissions (El-Gendy et al, 2003). Reliability, on the other hand, can be seen as the 'other-end' of packet loss as it is a measure of the number of packets successfully received by the source relative to those sent (El-Gendy et al, 2003).

All these are the most popular QoS metrics for conventional networked systems, other (usually more specific) metrics relating to areas such as availability and quantisation (El-Gendy et al, 2003).

A Survey of WSN QoS Metrics

So far, this literature review has looked at a wide-range of issues relating to wireless sensor networks and QoS. It has attempted to bridge the gap between the two areas by discussing the challenges WSN QoS faces and reviewing several previous attempts at implementing QoS in WSN's. It is important to recognise that just as the mechanisms for achieving QoS are not the same as those used for conventional networks, in many cases the definition of QoS often differs from that of conventional networks.

Before looking at the various definitions of QoS that have been used by previous researchers, it is important to remember that while conventional QoS mechanisms can be broken down into broad categories such as congestion avoidance (queuing and policing strategies) and management (queuing) the range of mechanisms used to achieve QoS in sensor networks ranges from routing protocols to middleware, mobile agents to MAC algorithms, and application software to network. Therefore it follows that unlike the fairly standardised field of conventional network QoS which has its popular metrics; wireless sensor QoS is likely to have various different definitions of QoS, depending on the motivation of the research or application of the intended system.

A logical question to ask at this point would be; despite the differences discussed earlier, are any traditional QoS metrics applicable to wireless sensor networks? The answer, like many things in this area, depends on the application. Younis and Akkaya (2004) state that because early wireless sensor networks were not intended for handling real-time traffic and because of their limited power, most early research focussed on energy-efficiency rather than QoS. However, they argue that as wireless sensor technology has progressed throughout the years, they have been given more advanced sensing devices that capture sound and video which has introduced different demands on sensor networks (Younis and Akkaya, 2004).

This has lead to a need for hybrid QoS systems that allow for the incorporation of multimedia QoS standards as well as power-saving technologies (Younis and Akkaya, 2004). However, although they accept the need for adaptation of traditional QoS in order to make it suitable for WSN's, they still focus on issues such as bandwidth and delay which are only really relevant to real-time multimedia sensor networks. Chen and Varshney (2004) argue that issues such as delay variation (jitter) are only really relevant to networks that contain a high-degree of multimedia traffic, and typical wireless sensor networks do not. As (at least some) traditional QoS metrics are inappropriate for use in conventional wireless sensor network applications, they argue that this means new ones that specifically take into account both the constraints and uses of wireless sensor networks must be derived.

An example of such a metric is that of packets per epoch used by Biyu et al (2005) which measures the number of packets received by the cluster-head in a period, this value is then used by cluster-head to determine a control value to be given to all active nodes, which they then use to determine when they will next transmit data. Although the packets per epoch value seems effective in the context of this system, it is not really that good a metric to be used. Monowar et al (2008) propose an interesting variation in which their system also varies nodes sending rates in order to achieve a desired level of QoS, however their work does not just look at one factor (i.e. packets sent in a period of time) but instead use a metric called the per-hop deadline miss ratio which is used to alter nodes sending rates. The per-hop deadline miss ratio is defined as a deadline miss ratio of all packets that pass through a node; this is used as a means of measuring delay and congestion, which are then used to adjust sending parameters (Monowar et al, 2008).

What this suggests, then, is that the approach of Biyu et al (2008) isn't necessarily a bad approach, but the packets per epoch metric is an inadequate measurement of QoS, as a low number of packets being received, does not necessarily indicate a high level of QoS if the low number is due to unreliable links rather than nodes efficiently managing their feedback schedule. The system used by Monowar et al (2008) is somewhat similar, however what distinguishes it

from the work of Biyu et al (2008) is the metric which incorporates deadlines, and therefore acts as a means of determining both delay and congestion.

Monowar et al (2008) are not the only researchers to feature some element of what is often referred to as congestion control. Congestion control usually takes into account the number of packets being sent by nodes but does not use it as the only measure of QoS as Biyu et al (2008) did. Many of these approaches are similar to that of Monowar et al (2008), in the sense that they equate QoS with the amount of time taken for a message to be received. This is similar to the traditional QoS metric of end-to-end delay, and some researchers such as Rahman et al (2008), believe that despite the differences between conventional networks and WSN's, end-to-end delay is a suitable metric for WSN QoS.

It is important to note, that in the case of Rahman et al (2008) that their work concerns a scenario dealing with real-time traffic, therefore delay is just as important (or possibly even more important) to this type of system as it is to say a video conferencing application on a traditional network. The system designed by Rahman et al (2008) also takes into account other traditional QoS metrics such as service rate and reliability level, as these metrics are shared amongst nodes, but the emphasis is on end-to-end delay as the primary method of congestion control involves 'splitting' the traffic being sent amongst different routes to ease the congestion in a particular area and therefore reduce delay.

Another interesting feature presented by Rahman et al (2008) is that of priority differentiation, in which packets assigned a higher priority, will be processed quicker, and in the event of congestion occurring, are less likely to be dropped than lower priority packets. The concept of treating packets differently based on some form of priority is key to traditional QoS, however due to the differences between WSN's and traditional networks typically the prioritisation used in WSN QoS will differ from that of traditional QoS.

As an alternative to the approaches seen so far in which systems are designed based on specific, pre-defined metrics, the work of Yaghmaee and Adjeroh (2009) involves the adaptation of the DiffServ marking standard for use in multimedia sensing applications. This means that instead of the QoS mechanism explicitly defining the type of QoS it will implement, this choice is made by the application which explicitly defines what level of QoS it requires (Yaghmaee and Adjeroh, 2009). Like the work of Rahman et al (2008) this scheme involves different levels of priority the main differentiation being between real-time and non-real-time traffic, although the non-real-

time traffic class has three subcategories (the highest of which still receives lower priority than real-time traffic) (Yaghmaee and Adjeroh, 2009).

Peterson et al (2009) also use a prioritisation scheme based on differentiating between traffic types, however unlike the work of Rahman et al (2008) or Yaghmaee and Adjeroh (2009), their approach also applies different priorities to different nodes. The reasoning behind this is that in their solution's proposed application (monitoring a volcano) if a specific event of interest occurs, then information from nodes within this area will become more important than 'standard' information from other nodes (Peterson et al, 2009).

In addition to containing some element of prioritisation, the work of the previous three all feature some form of application defined QoS. Whereas the work of Monowar et al (2008), Biyu et al (2005), and Barbancho et al (2006) explicitly defined the QoS metrics themselves, the previous three solutions examined had a more open ended definition of QoS in which the application was able to prioritise certain parameters at the cost of others. It might seem obvious, then, that those approaches in which applications are able to select different priorities have an advantage over those with 'fixed' QoS. However, it should be noted that this introduces an additional complexity, both in terms of programming and implementation, and hence is not suitable for all applications.

Some approaches such as that of Peterson et al (2009) intend to incorporate their work into a comprehensive middleware and application solution in which the end user is able take control of QoS priorities (i.e. in the volcano example the user could manually select a specific area to prioritise, or decide what data types are more important than others). However, it is important to emphasise here that not all of approaches are equal, that is to say that while Peterson et al (2009) aspire to incorporate their QoS solution into a comprehensive solution for an intended purpose, other the work of others such as Biyu et al (2005) focuses more on the lower-level mechanisms for providing QoS.

Therefore, although the idea of having specific users (whether human or software) select the desired level of QoS (and thus choose which metrics are relevant) is a fundamentally sound idea, but is not necessarily practical. It has been mentioned earlier that wireless sensor networks are generally designed and deployed for a specific purpose (whether short term or long term) and this purpose (as well as the environment the nodes are being deployed in) plays a major role in determining how they will operate. By extension of this statement, surely it could be held that it

is also the purpose (or application) of the network that then decides the level of QoS, and as a result what specific metrics are appropriate?

For instance, a simple network collecting one type of data such as temperature and then relaying it to a base station is unlikely to require tight end-to-end delays, or traffic prioritisation, and would likely sacrifice these for improved energy efficiency and reliability. Thus metrics such as delay would be inappropriate for this system. On the other hand, an advanced target tracking system that detected the presence of an object and then relayed multimedia data about it to the base station would almost certainly require prioritisation of some sort, bandwidth efficiency, low delay, and high reliability – but only for certain types of traffic. Therefore, this sort of system would deem bandwidth, delay, jitter, and reliability as important metrics but would perhaps be prepared to sacrifice energy efficiency as well as reliability and delay for less critical traffic.

The focus of this project is on the investigation of the effect clustering and data aggregation has on WSN QoS. The primary means of testing this will involve simulation of the iHEED application and then evaluation of the results. In order to determine appropriate metrics, it is therefore important to consider what iHEED does, which is to sense periodically and then transmit this data to the base using clustering, data aggregation, and multi-hop technologies (Younis and Fahmy, 2005). Therefore, it is unlikely that metrics related to real-time delivery will be of relevance, although as its authors envisage it as being applied to the real-world (Younis and Fahmy, 2005), a high degree of reliability is desirable as are low energy consumption and long lifetime.

Literature Review Conclusion

Although no specific literature review objectives were defined, the general purpose of the literature review can be inferred from the project objectives. That is to say, the main aims of the literature were to research a variety of areas related to wireless sensor networks, then narrow the focus onto WSN QoS, before going to look at two possible technologies for achieving this (clustering and data aggregation).

Once this was complete, the project began moving closer to the practical research stage by looking at the iHEED application, which is the focus of this project's investigations. With both a general knowledge of WSN QoS and the iHEED application, the final stage of the literature review looked at how WSN QoS was defined and from this several general metrics were drawn; high reliability, low power consumption, and long lifetime.

Using the work of the literature review, it is now time to prepare for the practical research stae of the project, and in doing so three hypotheses as well as a general list of aims for the practical research stage have been devised.

Hypotheses

The reason that no hypotheses were contained in the interim report was because despite the lengthy literature review that had already been undertaken it was decided that the project had not yet reached a stage where the implementation and testing portion was sufficiently detailed in order to determine whether an ostensibly relevant and productive hypothesis could be practically tested given the time and resources available.

On the other hand, it was important to ensure that hypotheses were developed before the formal experimentation stage begun, so as to ensure that they were the driving forces behind the experimentation and not the other way around. Although they have undergone some minor changes, mostly relating to their precise wording, the general aims behind the hypotheses have remained unchanged since their inception and their use subsequent use in defining testing and evaluation criteria.

Hypothesis One: clustering and data aggregation will ensure more efficient transmission of sensor readings without negatively effecting reliability

Hypothesis Two: through data aggregation and clustering, the average cost of sending and receiving a message (as expressed in energy consumed by the radio) will be reduced as will the average overall energy consumption

Hypothesis Three: despite the increased strain on cluster-heads, the reduction in traffic, and periodic re-clustering will not significantly decrease overall network lifetime

Primary Research Objectives

Objective One: Devise an Initial Testing and Evaluation Strategy:

With the literature review completed, the project will not move into the primary research stage. Before the implementation or testing stages can begin, it is necessary that the testing and evaluation strategy is completed. Initial work at this stage will focus on making the transition between the literature review and primary research stages by making use of the knowledge acquired from the literature review to build an initial test plan.

This initial plan will most likely be high-level in detail with the emphasis being placed on formalising the QoS requirements and beginning to question how these can be tested and evaluated.

Objective Two: Finalise the Testing and Evaluation Strategy

As the testing and evaluation strategy is worked on, it will progressively become more technical in detail, however it is important to ensure that the strategy is still focussed on testing the correct metrics as there is no point in devising sophisticated tests and analytical processes for the results if the data being captured is irrelevant to the overall aims and objectives of the project. Work will also begin on determining how the iHEED application should be implemented and what modifications must be made in order for it to be simulated using TOSSIM.

Objective Three: implement the iHEED application and ensure it is functioning correctly

With the testing and evaluation strategy and implementation completed, the implementation phase should go ahead according to the implementation plan. This stage will involve modifying TinyOS in order to run the iHEED application (and possibly modifying the iHEED application as well) and it is important that the implementation plan is adhered to all points in order to provide a logical sequence of events from start to finish.

Initial testing will also be performed at this stage, however it is important to emphasise that this type of testing will focus on determining whether the iHEED application has been correctly altered, and if not, previous work will be reviewed in order to determine what has gone wrong and how it can be rectified.

Objective Four: test iHEED via simulation

With iHEED implemented correctly the next stage is to conduct testing in the form of simulation. Simulation will take place using TinyOS's TOSSIM simulator (or the PowerTOSSIM plugin for more detailed energy analysis) based on the testing and evaluation strategy. Again, it is important when programming the simulation scenarios that the scenarios are suitable and the correct metrics are gathered, as if the wrong the data is collected at this stage then the evaluations will be flawed, and as a result the entire project will have been a waste of time.

Therefore, it is important to 'test the test' so to speak, by choosing metrics and performing simulations with a small number of nodes over a short period of time to ensure that the simulations are functioning correctly and the correct data is being collected.

Methodology

Nature of the Experiment

The practical research portion of this project can be best classed as primarily simulation based. The primary aim of this project is to evaluate the effectiveness of clustering and data aggregation as means of managing WSN QoS.

In order to achieve this aim, research will focus on the iHEED application, which as mentioned before, is a TinyOS application that combines clustering and data aggregation with multi-hop routing. In order to evaluate what effect the iHEED application has had, it will first be modified in order to be simulated and then several tests will be run to collect various metrics related to the hypotheses discussed earlier.

As a means of comparison, the original Surge application, of which iHEED is a modification of (Younis and Fahmy, 2005) will be put through exactly the same simulations, and the results compared in order to determine whether or not the modifications made to iHEED have resulted I an improved level of QoS.

The metrics that are to be collected as well as more detailed aspects pertaining to the configuration of the simulation will be discussed in subsequent sections, however the purpose of this section and its sub-sections is to provide a broad overview of the iHEED application and some general issues relating to this experiment.

At this stage, it is important to emphasise that despite the classification of this research as primarily simulation based, there still exists several crucial differences between this experiment and traditional network simulation experiments. The most obvious of these is that although a dedicated simulator (TOSSIM/PowerTOSSIM) is being used, unlike 'traditional' network simulators such as NS2, OMNeT++, or OPNET which are all 'stand-alone' applications TOSSIM/PowerTOSSIM is in fact a part of the TinyOS platform.

Therefore, in order to make use of the TOSSIM simulator the entire TinyOS operating system must be installed. Without going into the mechanics of TOSSIM simulation (this will be

discussed later on), it is pertinent to mention that very little of the simulation configuration is actually managed by TOSSIM. In TinyOS, if the user wishes to simulate nodes running a particular protocol, then there must be an existing TinyOS application for it (or they can program one themselves), and if they then wish to configure that algorithm (i.e. set the frequency that update messages are sent) then they must do so by altering the application rather than configuring this in the simulator.

Similarly, most of the data collected from TOSSIM simulations is recorded through the use of debugging statements which are inserted into application and system code to print messages when a particular event occurs (i.e. a packet is transmitted, a component fails, etc.) which is in contrast to traditional network simulators where the user is able to select what metrics they wish to be recorded, and often view the output in a graphical manner.

As subsequent sections will reveal, after suitable metrics for this project were drawn up, it was apparent that not all of them could be collected through simulation using the pre-defined debug messages. Therefore, it was necessary to design and implement custom debug statements for the purpose of gathering specific metrics. In addition to this it was found, after early tests, the the iHEED application would require modification in order to be simulated. Naturally this brings up the issue of modifying one of the the test subjects, and this and other potential complications are discussed in the Implementation sub-section.

Therefore, despite the project being classified as simulation-based experimental project, it remains a somewhat unconventional simulation-based project, as the implementation and use of debug statements for testing actually bears more in common with a develop and test project. This is not entirely surprising, both due to the novelty of the means of testing (TOSSIM and TinyOS rather than a conventional stand-alone simulator) and the fact that this project was originally conceived as a develop and test project in which the aim was to modify the iHEED application.

Unfortunately, due to time and resource constraints, it was decided that such an aim was infeasible and the focus of the project moved from modifying the iHEED application to testing and evaluation it. However, it is hoped that the results of the simulation and subsequent analysis will be of use in identifying future augmentations for the iHEED application in particular, and the HEED protocol in general.

Why Simulate?

Given that a large part of the previous section details why the TOSSIM simulator is unlike a conventional simulator, and why the iHEED application must be modified in order to allow for simulation, why simulate seems like a very relevant question. Although many researchers such as Peterson et al (2009) argue that simulation provides a significantly lower degree of accuracy than testing on actual nodes, there are several counter-arguments in favour of simulation.

Merret et al (2009) go so far as to claim that simulation is in fact the most popular means of for testing new WSN technologies. The reasons that are given for the popularity of simulation include the reduction of time, cost and effort as a direct product of using a software environment that allows users to perform in-depth debugging and collection and analysis of results (Merret et al, 2009). Eriksson (2009) expands on this by saying that the accuracy of simulation-based debugging methods is higher than testbed debugging as simulation software often includes explicit support for debugging.

The only way that conventional debugging could be carried out on actual nodes would be to either have nodes store debug messages to their (extremely limited) flash memory or send them to another device (which would require a direct connection from each node to the recording device). As these approaches are extremely impractical, they are rarely utilised, to the extent that TinyOS does not run debug code on actual nodes but only in simulations.

As will be explained later on, some of the metrics being collected for this project involve the extensive use of debug statements. The metrics in question are related to efficiency and reliability, the latter represents the number of packets sent to the base against the number of packets received and was in fact used as a metric for the initial testbed evaluation of iHEED (Younis and Fahmy, 2005). In the testbed evaluations this metric (referred to as successful transmissions) was collected by analysing the packets received by the base node to determine the number of sensor readings it contains (i.e. how many the cluster-head has sent to the base node) although this approach does not appear to take into account the number of packets sent by cluster members to cluster-head and thus does not offer a full view of transmission reliability.

In this project, the reliability metric is collected by using debug statements to record when an ordinary node sends a message, when a cluster-head receives it, when a cluster-head sends the combined readings, and when the base station receives these. This allows for a far more accurate

view of reliability at all stages of the network. However, as this method relies solely on the use of debug statements (due to the lack of viable alternatives) it would be impossible to implement outside of simulation, and therefore serves as an example of why simulation allows for greater flexibility in evaluation sensor networks.

That said it is important to bear in mind the fact that despite its popularity and flexibility, simulation is still not as accurate as testbed evaluation (Eriksson, 2009). Shnayeder (2005) argues that despite the reduced accuracy simulations are more scalable and allow for a greater number of nodes to be examined than would be practical using a testbed evaluation. Therefore, given the compelling cases of both sides it is seemingly apparent that the best approach would be to simulate and debug the system before deploying it on a small number of actual nodes in order to evaluate how it performs on actual devices.

Unfortunately, due to limited time and resources it is not possible to utilise both approaches for this project. However, given both the compelling arguments for simulation as well as the fact that the original iHEED application has already been used in a testbed evaluation, for the purposes of this project, simulation will be the means of testing and evaluating the iHEED application.

It has already been mentioned that the focus of the practical element of this project will be on the simulation and subsequent evaluation of the iHEED application, however, an important aspect has been overlooked up until now; how exactly will iHEED be evaluated?

The most obvious answer to that question is via some form of comparison, which itself brings up the question of what should iHEED be compared with? Official releases of TinyOS contain no clustering or data aggregation applications, indeed iHEED itself is an 'unofficial' application that has been developed by third parties, and according to a search of literature and electronic sources, is the only widely-available application of its sort.

This might seem like a considerable problem, however, as mentioned earlier, the iHEED application is primarily based on modifications to the Surge application available in TinyOS 1.x (Younis and Fahmy, 2005). Surge is an application that has been designed to demonstrate multi-hop routing, by having nodes collect data and relay it to the base node in a multi-hop fashion (Malesci and Madden, 2006).

TOSSIM and PowerTOSSIM

The simplest method of carrying out simulations in TinyOS is by using the TOSSIM simulator that comes with all versions of TinyOS (Levis and Lee, 2003). TOSSIM is a discrete event simulator that allows users to conduct simulations of TinyOS-based WSNs in order to test applications, perform debugging, evaluate performances, or simply learn more about a system (Levis and Lee, 2003).

An interesting feature of TOSSIM that sets it aside from other network simulators is its modelling the interaction of devices at network bit rather than packet level (Karl, 2005). This is accomplished through detailed abstractions of device hardware (Karl, 2005).

Although TOSSIM provides highly-detailed simulation of nodes running TinyOS, one major problem is that it does not come with analysis of node's power consumption (Shnayder *et al*, 2004). This is a big drawback, because as described in the literature review, almost all nodes are battery powered and thus have limited energy.

The lack of energy modelling is one of the reasons that (Shnayder *et al* (2004) proposed PowerTOSSIM, which is a plugin for TOSSIM that allows for detailed recording and analysis of nodes energy consumption. Therefore, for the parts of this project which require energy analysis, the PowerTOSSIM plugin will be used.

Proposed Metrics

One of the major problems in defining WSN QoS metrics is that typically no two real-life WSNs are identical. Even using the same platform (hardware and software) it is likely that there will still be some major differences between the two networks. One significant difference is application, that is to say what the network is being used for.

A network consisting of 2000 Mica2 nodes running TinyOS could be deployed for object tracking or environment monitoring. Exactly the same hardware and operating system is being used, but the difference in application is significant.

Applications such as object tracking, or real-time monitoring (i.e. in security or safety-critical systems) are likely to place tight constraints on timeliness at the cost of other metrics (i.e. energy used, congestion, etc.) while other systems such as environment monitoring are probably more likely to emphasise reliability and energy conservation.

The two applications being evaluated both fall into the latter category as the purpose is to sense (photographic) data and then relay this to the base. Therefore, with both the nature of the applications and the aims of the project in mind, the following groups of metrics have been defined.

Efficiency and Reliability

These two metrics are aimed at directly testing the first hypothesis which states:

Clustering and data aggregation will ensure more efficient transmission of sensor readings without negatively effecting reliability

It is clear from reading this that there are two parts to it; the first involves the efficient transmission of sensor readings. What this means is that instead of constantly taking and sending readings, which will result in a great deal of strain on both the network in general and the base node in particular, the use of clustering and aggregation will reduce the number of readings being sent. An immediate problem with this is that due to quirks of sensor networking, it is possible that the base station could receive a reduced number of sensor readings (when compared to the performance of Surge) yet this could simply be down to unreliable transmission rather than efficiency.

This links into the second part of the hypothesis which states that the (seemingly) increased efficiency will not be as a result of reduced reliability. This entails then that some form of reliability will be measured, both for the sake of gauging reliability as well as being used to ensure that any reduction in transmissions is as a result of increased efficiency rather than decreased reliability.

Efficiency

This metric is used as a determination of how many packets are received by the base node at the end of the simulation. The purpose of this metric is to determine what effect data aggregation has had on the number of packets that the source node receives compared with the results from the original Surge application.

In the case of this project, the first method was used to determine the number of application-related packets (i.e. sensed data) while the second was used to ascertain the total (i.e. both application and control) packets received by the source. The reason that two different measurements were taken was to allow for the determination of what effect the Surge/iHEED application has had on the traffic.

This can be measured by either manually counting the number of packets the source node has received at the end of the simulation (i.e. by adding debug statements to record this), or using a counter to specifically keep track of this (such as the one contained in the Serial Forwarder application).

A similar metric is the 'packets per epoch' used by Biyu et al (2005), which as its name suggests, measures the number of packets sent in each period (epoch). The fundamental differences between this metric and the efficiency metric is that the packets per epoch focuses on packets being sent, rather than received, and splits the total experiment time into smaller periods rather than looking at the life of the system as a whole (Biyu et al, 2005).

In addition to these differences, another important aspect of the efficiency metric is that despite what was mentioned earlier, it does not only take into account the total packets (both in terms of application and control) received by the base node, but it also takes into account the reliability of the network (with regards to application traffic).

Without the reliability metric, the efficiency metric is somewhat incomplete, as it would not take into account how many packets were sent in the first place, therefore using the efficiency metric on its own could lead to false conclusions; i.e. if the results of iHEED showed the base node

received less packets than with Surge this could be down to either efficiency of data aggregation, unreliable communications, or a combination of the two. Therefore, it is important that efficiency covers more than just the total packets received by the base.

Reliability

As discussed in the literature review, reliability is a popular metric for traditional and ad-hoc networks and serves as a measure of packets sent against packets received. Although reliability is perceived as being an end-to-end metric, and as wireless sensor networks very rarely offer end-to-end connectivity, it has been argued that reliability is still important to sensor networks, albeit not in the traditional end-to-end manner (Sankarasubramaniam et al, 2003).

Therefore, although end-to-end connectivity rarely exists, there would be little purpose in having a sensor network if a node was unable to relay data to the base node as the node would merely be sensing and sending, but the user would never receive this data. While this is an obviously extreme example, a sensor network in which a small number of readings reach the base is undesirable; therefore to sensor networks reliability is still important.

Although the revised model proposed by Sankarasubramaniam et al (2003) is incompatible with the work of this project as it focuses on event-based systems (i.e. those in which nodes send data to the sink in response to a particular event occurring) it is still possible to do as they did, and use a non-end-to-end form of reliability. This form of reliability will involve nodes recording (via the use of debug messages) the number of sensor readings they have sent, which will then be compared against the number of sensor readings the base station receives. As this approach only takes into account the reliability of specific types of traffic (i.e. sensor readings sent to the base node) it reduces the number of packets being measured (by not recording the large number of routing traffic being sent between nodes) while avoiding the mistake of viewing reliability in end-to-end terms (i.e. it does not matter if a node can reach every other node in the network so long as they are able to send their readings to the base).

Typically this would be calculated by comparing the percentage of packets sent to the base node against the number received. In the unmodified Surge application this would be as simple as comparing the total number of packets sent to the base against the total received. However, as

iHEED aggregates packets along the tree, this would be harder to investigate, although not impossible.

For iHEED reliability is split into two phases; node to cluster-head and cluster-head to base, therefore bypassing the problem of aggregation skewing reliability. The two-stage approach used with iHEED also allows for a more accurate picture of node to cluster-head and cluster-head to base communications that could be of use in future research.

It has already been demonstrated how the efficiency metric is dependent on the reliability metric to give an accurate determination of efficiency, however, it is also important to establish that the reliability metric is itself dependent on the efficiency metric. Both the reliability and efficiency metrics share a common element; packets received. Although, calculations for reliability of iHEED include two measures of packets received (at cluster-head and base) only the latter of these will be used for calculating efficiency.

Energy Consumption

Although the group of metrics, in dealing with efficiency and reliability was directly linked to the first hypothesis, it is still somewhat relevant to these metrics which are based on the second hypothesis as listed below:

Hypothesis Two: through data aggregation and clustering, the average cost of sending and receiving a message (as expressed in energy consumed by the radio) will be reduced as will the average overall energy consumption

The reason for this statement is that the efficiency metric relates to the number of packets sent by ordinary nodes and received by the base station, therefore if the number of packets being sent by nodes and received by the base is decreased this will present a saving in terms of energy required to send/receive and process the messages. However, this does not necessarily mean that the energy used by the radio or other components will be reduced overall. Through the use of clustering iHEED requires cluster-heads to receive messages from all of their members which they will then transmit to the base node.

Obviously this means they will be using their radio more than ordinary nodes, however as they are aggregating all of their received messages into one packet the cost of sending will be lower than if they were merely relaying the packets unmodified. Similarly, the use of cluster-head rotation strategies prevents nodes from being cluster-heads for too long a period of time.

In order to measure the impact that clustering, data aggregation and the other modifications iHEED contains, have had on power consumption overall, two primary metrics have been devised; cost of transmission and overall energy consumption, both of which contain totals and averages.

Cost of Transmission

This metric represents the total amount of energy used for transmitting data. The purpose of this metric is to determine what effect iHEED's use of clustering and aggregation has had on the average amount of energy being spent sending messages. As cluster members send their messages to cluster-heads who form a tree with other cluster-heads and continually aggregate this data along the tree, the distance which ordinary nodes are sending should be reduced. Similarly, re-clustering reduces the chance of nodes remaining cluster-heads for the duration of this experiment.

Therefore, it is important to determine what effect these measures have had in terms of the amount of power consumed by sending messages. This data can be obtained in both cases by simulating using PowerTOSSIM and analysing the data using the post process tool.

Overall Energy Consumption:

As its name suggests, this metric represents the average overall energy consumed by all nodes in the network. This metric will also be used in conjunction with the average cost of transmission metric, in order to determine what effect the sending of messages has had on the overall energy consumption.

This metric will be measured by using the Post Process analysis tool with PowerTOSSIM and then totalling the results.

Network Lifetime

Just as the previous group of metrics was related to the first via their hypotheses, this metric has been designed so as to answer hypothesis 3 and collect information on how iHEED's modifications affect its lifetime.

Hypothesis Three: despite the increased strain on cluster-heads, the reduction in traffic, and periodic re-clustering will not significantly decrease overall network lifetime

In WSN's, network lifetime is typically defined as either the time between the network commencing operation and either the first or last node dying. Due to the nature of the application (a simple sensing and data-relay exercise), the second of these two metrics is probably the most appropriate, as the loss of a single node should not dramatically affect the outcome. More complex systems might set a minimum percentage of active nodes, and when the level falls below this the network is of no use, or a period of time for which a certain number of nodes should be active.

This metric should be easy to obtain as the node death shows up in TOSSIM, although a potential problem would be if the set simulation time was shorter that the network lifetime in which case at least one node would remain fully operational at the end of the simulation. A solution to this would be to run simulations without a set time, specifically for the purpose of determining network lifetime.

Testing Plan

The Testing and Evaluation section was intended as a higher-level discussion of the metrics being used for testing and evaluation. Building on this work, the purpose of this section is to break the testing stage of the project down into various steps and explain what will happen at each stage and what the intended outcomes are.

Breakdown of Testing Activities

Task 1: verify correct configuration of test environment

This is really just a preliminary measure as previous stages of this project have already dealt with the issue of ensuring that iHEED has been implemented correctly. As explained in the previous section, two instances of the XubunTOS virtual machine have been created; one containing the iHEED application and the other containing an unmodified version of TinyOS 1.x.

This stage will focus on confirming that both versions are correctly configured by conducting various basic simulations to ensure correct performance.

Task 2: perform reliability and efficiency simulations

The first of the metrics to be simulated are efficiency and reliability. Despite their differing complexities, both metrics can be collected from the same simulations as the means by which they will be captured (TinyOS debug messages, described in detail later) allows for all of the relevant statistics to be collected by running each application once.

Task 2: perform energy simulations

The reason that these metrics are being tested second is for simplicity; all of the data can be gathered using PowerTOSSIM and the Post Process analysis tool. This information can then be saved as a text file (one each for the Surge and iHEED simulations) which can be analysed at leisure.

Task 4: calculate network lifetime

This is probably one of the most simple metrics to collect, however it is undoubtedly the most time and resource intensive. Whereas the previous metrics were collected over a predetermined period of time, the very nature of this metric prohibits such a configuration. As the purpose of

this metric is to gauge how long the network can survive, the simulation will be configured without a run-time so that it can instead run until the last node is dead.

The obvious product of this experiment is the time at which the last node dies, however, as described previously, the way in which this data will be collected allows for more complex calculations to be made.

Task 5: verify correctness of collected data

This stage is perhaps somewhat vague, as the exact steps taken will vary depending on the nature of the data. In general terms, the purpose is to collect and aggregate the data from multiple runs of the same experiment in order to ensure that there are no obvious anomalies due to incorrect configuration or other errors.

Task 6: analyse the data numerically

The final step in the testing stage involves performing the necessary calculations and analysis on the data in order to ensure that it is in a format suitable for evaluating.

Implementation

Whereas Surge comes as standard with TinyOS 1.x, and is able to run on both actual nodes and TOSSIM without any modification, Surge is not intended for simulation and will thus require several modifications to be made before it can be simulated.

iHEED

The iHEED project page states that the iHEED TinyOS application has been tested on on Mica2 and Mica2Dot sensor devices (Younis and Fahmy, 2005) however there is no official implementation for the TOSSIM simulator. This is potentially problematic as this project relies on the ability to simulate the iHEED application using TOSSIM, attempts to compile the application without modification resulted in several errors, as the output from the command line in Appendix 2 shows.

On further inspection of the error messages, it is apparent that several of the errors are in fact due to a missing '/' at the start of 'MultiHopEngineM.n' which is easily rectified. However, the errors given in relation to 'MultiHopLEPSM.nc' and 'MultiHopRouter.nc' relate to the use of the interface 'CC1000Control'. A quick look at the TinyOS manual reveals that this refers to the 'CC1000 Radio Stack' which represents the transceiver used by Mica2 sensing devices (TinyOS, 2003). This could be solved by altering the appropriate files to remove any references to the CC1000 device and replace them with the generic TOSSIM radio model, however this approach would sacrifice the accuracy of the results.

Another solution exists, and that is to make use of the PowerTOSSIM plugin designed by researchers at Harvard University to extend TOSSIM in order to allow for more detailed analysis of power consumption (Shnayder et al, 2004). One of the main advantages of using PowerTOSSIM as an alternative to TOSSIM is that PowerTOSSIM can model the CC1000 radio stack and therefore simulate it (Shnayder et al, 2004).

Although there is no formal (or indeed informal) documentation involving attempts to simulate iHEED using PowerTOSSIM it was decided that due to the opportunity to simulate a more realistic radio model as well as the fact that PowerTOSSIM was always intended for use in Experiment 2, it was decided that attempts to simulate iHEED using PowerTOSSIM should be made before considering modifying iHEED's radio model.

The PowerTOSSIM manual (Shnayder et al, 2005) states that in order to simulate the CC100 stack the following command must be included in iHEED's Makefile:

```
PFLAGS += -I%T/platform/pc/CC1000Radio
```

After adding this line and rebuilding the iHEED application, an initial simulation using just the 'route' debug option was performed which seemed to indicate the program was performing as intended. Following this initial test, iHEED was simulated using various other debug types (i.e. 'radio', 'clock', etc.) and no further problems were encountered until the simulation was performed using the 'sense' debug mode. The error returned was 'Error: Unable to establish EEPROM of correct size', a quick investigation revealed the source of this message to be the eeprom.c file located in the 'tinyos-1.x/tos/platform/pc' folder.

Reviewing the documentation available on the SourceForge repository revealed that file in question was used to model the flash memory (EEPROM) of devices being simulated. The documentation also revealed that the version of the epprom.c file being used contained modifications that allowed for more realistic modelling of nodes memory by having newly created EEPROM initialised with all 1's instead of 0's (see Appendix 3).

Further browsing of the repository revealed an earlier file that did not have this addition, and it was decided to use this file for simulation of the iHEED application. Although the use of a less-realistic EEPROM configuration may sound contrary to the goal of accurate simulation, it was decided that as iHEED had been extensively tested on actual nodes, despite the problems encountered with simulation it obviously hadn't caused any problems relating to the flash memory of the testbed nodes.

Therefore, the older version of the EEPROM initialisation was chosen (Appendix 4 shows older versions) to be used for both iHEED and Surge simulations. Although Surge had not posted any problems in relation to EEPROM initialisation, it was decided that in the interest of ensuring that conditions for both applications were as similar as possible it would be simulated with exactly the same EEPROM model as iHEED.

Once this problem had resolved, testing via simulation of all debug modes resumed with no further problems being identified. Following the completion of debug simulation, as with Surge, iHEED was simulated using the TinyViz application in order to gain a better understanding of how the application worked.

While running simulations with the 'radio links' option enabled, it was noticed that, unlike Surge, all communications were multicast only, rather than direct node-to-node. This seemed odd, as one of the main aspects of iHEED was the use of clustering in combination with multi-hop routing, therefore, it would seem logical that after an initial period of time (around x reported by Younnis and Fahmy (2005)) clusters would be formed with member nodes transmitting data to cluster-heads who would then send it to the base in a multi-hop fashion.

Despite the number of nodes and duration being altered, neither an increase in nodes (up to 1000 were simulated) or duration (simulation times of over an hour were used) all communications

seemed to be of a multicast nature. An initial investigation of iHEED's SurgeM.nc file (an annotated version of which is available in Appendix 6) did not reveal any obvious errors, however, it was when comparing iHEED's SurgeM.nc with the original SurgeM.nc file that a compilation error (as show in Appendix 2) was noticed.

As the code shows, iHEED's version uses the double instead of single equals, which means that when this is called a comparison rather than a value setting takes place. In addition to changing the equals, the debug statement; 'dbg(DBG_USR1, “SurgeM; Packet Sent \n”)' was inserted in order to quickly ascertain whether the modification had worked. Simulating with the debug mode set as usr1 revealed that the modification had indeed worked, and that iHEED was now sending sensor readings in the correct manner. This was further confirmed through TinyViz simulation which showed direct radio links appearing between nodes at around 100 seconds.

Simulation Set-Up

Simulation Environment

Two separate XubunTOS virtual machines will be used; the first containing the unmodified version of Surge, and the second containing the iHEED application. The reason for using two separate virtual machines is that in addition to containing modifications to the Surge application, iHEED also contains modifications to TinyOS system files such as those concerning multi-hop routing. Therefore, rather than try and switch between iHEED and the original Surge by adding or removing code, it was decided that using two completely different instances of the virtual machine was the most appropriate solution.

All simulations will be conducted using either the TOSSIM where energy consumption is being measured, the PowerTOSSIM.

It is also important to emphasise at this point that different metrics will be collected separately, that is to say that, one simulation run might collect delay and reliability, whereas another might collect the various energy statistics. The reason for this is due to the constraints of TOSSIM/PowerTOSSIM which lack many of the features contained in various general purpose commercial (i.e. OPNET) or open source (i.e. OMNeT++) simulators such as a GUI, advanced metric collection/simulation configuration options, etc.

As discussed earlier, both TOSSIM and PowerTOSSIM are programmed from the command line using simple commands and extensions, the approach is to emulate actual node performance with the aim of testing and debugging applications before deploying them on a testbed. Therefore, it is important to appreciate that unlike GUI-oriented simulators such as OPNET or OMNeT++ which both contain numerous options for configuring nodes, simulations, and metrics most of the information gleaned from TinyOS simulations is via the use of debug statements. In particular, the efficiency and reliability metrics both make use of debug statements that have been added to their original code purely for the purpose of collecting these metrics.

As such this section tends to focus more on such low level configurations and therefore contains several references to source code (relevant sections of which are included as appendices) than high-level screenshots of configuration GUIs.

Simulation Configuration

Of the literature reviewed previously, it was noticed that the majority of sources did not explicitly state the duration of simulated experiments and those that did featured very short durations. For instance, Choe et al (2009) used 100 seconds as their duration whereas Rahman et al (2008) used an even shorter duration of 60 seconds.

While wireless sensor networks are typically only employed for short periods of time, these times are probably too short for simulating iHEED accurately as the parameters used by Younis and Fahmy (2005) would mean that clustering took around 77 seconds to complete.

Therefore, it was decided that more literature needed to be reviewed in order to gain a wider knowledge of WSN simulation. The work of Manjeshwar and Agrawal (2002) is based on the development of a routing protocol called APTEEN, with the protocol being tested by in simulations of 100 nodes over the period of 2500 seconds (just under 42 minutes).

Throughout this paper, the importance that application has on WSN's has repeatedly been emphasised. Therefore, when considering simulation parameters (both in terms of simulation duration and node quantity) it is important to not only consider what parameters have been used

for previous research, but what parameters are realistic representations of real-life WSN applications.

It is also important to consider time and resources needed for performing such simulations. This means that another important factor to consider is the data that is being collected as well as the number of nodes being simulated, therefore, rather than having a set duration or number of nodes, these parameters will vary between experiments.

As mentioned earlier, the most appropriate simulation duration for most parameters might not be appropriate for measuring network lifetime as it is possible that by the time the simulation has finished some (or all) of the nodes may remain active. Therefore, in order to accurately gauge network lifetime, a separate simulation of both applications will be performed in which simulation duration is not configured, therefore the simulation will run until the last node dies.

Setting parameters such as time and number of nodes is fairly simple, as with most elements of TOSSIM simulations, this is configured from the command line for all of the experiments except experiment 3, after setting the appropriate debugging options the following line would be issued; 'build/pc/main.exe [number of nodes] [duration] > outputfile.txt' where the parameters in square brackets would be substituted with the appropriate values and the format for the output file would be the name of the application (either Surge or iHEED) followed by the experiment number (1, 2, or 3).

Experiment-Specific Configurations

Whereas the previous section dealt with the general simulation environment (i.e. VMWare virtual machine's) and configurations (times, number of nodes, etc.) the purpose of this section is to deal with the more specific details of each of the three main experiments.

Experiment One – Efficiency and Reliability

Although it may have appear as though these are two separate metrics, they are in fact both inter-linked and reliability is calculated based on part of the efficiency metric which can be broken down into two main sub-metrics; total readings received and total packets received. As explained

in the metrics section, the former is a measurement of how many sensor readings the base station receives from member nodes while the later represents the entire volume of traffic received at the base station (i.e. both sensor readings and control packets such as routing updates).

The number of nodes being simulated is 500 and the duration will be 30 minutes. Initially it was hoped that 1000 nodes would be simulated for an hour, over this proved unrealistic and took nearly two days to complete! Therefore, the number of nodes and duration is something of a compromise, but should still allow for an accurate picture of efficiency and reliability to be formed.

Efficiency:

Of these metrics the former is the easiest to collect, as it merely involves connecting the Serial Forwarder application to the base station (obviously this is done 'virtually' as no physical nodes are used for these experiments) and observing the value the packets received counter depicts at the end of the simulation.

The second efficiency sub-metric is slightly harder to collect due to the need to differentiate between sensor readings and other types of packets received at the base node. An ostensibly simple solution to this would be to run the simulation with the 'Active Messages' debugging option enabled which would for the recording of all packets sent and received by the networks nodes. The obvious problem with this approach is the sheer volume of packets (sensor readings, routing updates, acknowledgements, etc.) that would potentially appear twice (i.e. being sent and received) in a network consisting of 1000 nodes being simulated for one hour.

Therefore, while this approach is theoretically workable, it is not desirable and so a more efficient approach was sought. This approach turned out to involve the specific inclusion of debugging messages in the Surge application code so that the base station would print a simple message “BaseStation: Received Sensor Readings” every time it received a Surge packet from another node.

This approach proved relatively easy to implement in iHEED which included an 'InterceptSurgeMsg' event as show in Appendix 6).

This event was modified slightly to include the command 'dbg(DBG_USR3, “BaseStation: Received Sensor Readings”);' after using an if statement to verify that the node was the base station and the packet received was a Surge message.

If the if statement used to identify the node as the base station and the message type as Sensor readings looks familiar then this is because it has been copied verbatim from the else# clause which defines behaviour for intercepting messages without clustering being applied.

The original Surge application did not contain the 'InterceptSurgeMsg' and a quick survey of appropriate file (contained in Appendix 5) shows that no equivalent event is available. Therefore, after some consideration of alternative options, it was decided that a modified version of the 'InterceptSurgeMsg' event from iHEED should be included. The code for this is shown below

Naturally this version does not contain any functions related to clustering or iHEED's energy management system and only contains the bare minimum required to perform interception, verify the intercepting node is the base station and that the intercepted packet is sensor readings and then print out the appropriate message.

Naturally the inclusion of this new event (and the modification of the original event in iHEED) brings into question the issue of modifying the original protocol, however, it is important to remember that the other method available would most likely require the creation of a dedicated application in order to perform the appropriate analysis of Active Message results, while the actual simulation would last for a considerable duration to the large number of Active Messages being simulated.

Reliability:

As stated several times in this report, the reliability metric is dependent on the total readings received portion of the efficiency metric as a means of ascertaining how many sensor readings the base receives, and combining this with the total sent allows for the number of successful transmissions to be gauged.

In the original Surge application collecting the total number of packet sent is simply a matter of using an if statement to determine if the node is the base station inserting this piece of code; dbg(DBG_USR3, “Sending Sensed Data \n”);' after Surge sends a message.

The result of this addition is that when a node has sent its readings it will first check to see if it is the base node, and if it isn't it will print the message 'Sending Sensed Readings'. The reason for the if statement preceding the debug statement is that like normal nodes the base station also transmits sensor readings, however the base station transmits these to an observing computer and not via the WSN, therefore, having the base station print this statement would render the statistics incorrect.

As with the total readings received sub-metric, the results will be analysed using the Grep search facility to record the total number of 'Sending Sensor Readings' statements that were printed and this will be compared with the number of 'BaseStation: Received Readings'.

Unfortunately, collecting this sub-metric it is not really this simple for the iHEED application as cluster-heads combine the readings of several nodes into one packet which they then transmit to the base station, thus even if every transmission was successful, the number of packets sent by the nodes would still be more than the number received by the base.

Therefore, it was decided that packets being sent in the iHEED application should be measured in two stages; the first is the sending of the packet from an ordinary node to their cluster-head and the second is the cluster-head sending the aggregated packet to the base node.

The first of these is recorded in a manner similar to the way messages sent are recorded for Surge; a debug statement that prints the message 'CLUSTERMEMBER: sending sensed data to cluster' is then added after the node sends data.

The second stage (i.e. cluster-head to base node) is recorded by having cluster-heads print the message 'CLUSTERHEAD: sending readings' once they have sent their aggregated readings to the base.

For packets received by the base node, as with Surge, this information is obtained from the efficiency metric, while packets received by the cluster-head are recorded by using a debug statement to print 'CLUSTERHEAD: received sensor readings' in the interception portion.

Once the simulation is complete, the efficiency metric will be calculated first, and following on from this the total number of packets sent by cluster members and received by their cluster-heads will be calculated in order to determine the first portion of the reliability metric. Following on from this, the number of aggregated packets sent by cluster-heads will be obtained and used to determine the cluster-head to base station reliability.

Experiment Two – Energy Consumption

Technically this involves the collection of several metrics (and sub-metrics), however as all of these metrics will be collected from the one simulation they have been grouped together for the one experiment. This experiment relies heavily on the functionality of the PowerTOSSIM simulator and it's Post Process tool (both of which were previously described in detail) in order to collect both the raw energy usage data (PowerTOSSIM) and then analyse it to obtain total enrgy usage a break-down of component energy consumption statistics.

Unlike Experiment 1 both the number of nodes and duration will be shorter, with 100 nodes being simulated over the period of ten minutes. This is again due to time constraints, as simulating using PowerTOSSIM means that every activity that consumes power is recorded, thus increasing the simulation time dramatically.

Another difference between this experiment and Experiment 1 is the debug mode will be set to power (using the command 'export DEBUG=power') and the simulation command will include the '-p' flag ('build/pc/main.exe -p -t=600 100 > power.txt') in order to inform TOSSIM that it is using the PowerTOSSIM extension and recording debug messages related to power consumption.

Once the simulation has been completed, the 'power.txt' (which for actual simulations will be prefixed with either 'surge' or 'iheed') will then be analysed by the Post Process command:

Which will analyse the raw power consumption data and output a detailed breakdown of each node's energy usage over the duration of the simulation

Experiment Three – Network Lifetime

As stated earlier, this may seem like one of the simplest metrics to capture, although it is important to note that the analysis of the metrics can be more complicated, however such analysis largely falls under the category of evaluation and will be discussed in the appropriate sub-section.

Indeed, the method of capturing this metric merely requires that before the simulation is run the debugging status such as 'Radio', 'Clock', or 'Power'. Of the various debug modes available for use in TinyOS, these three are the only ones that display information related to time (i.e. Radio will print when a node sends or receives a message), and as the means of estimating a node's lifetime is by determining when it last performed an activity, using a debug mode that prints times is essential.

Of these nodes three eligible modes, it was decided that Power should be used for determining node lifetime, for the reason that during simple tests it was found to perform simulations at the fastest time. Due to the need to simulate the network for an unknown period of time, it has been decided to again decrease the number of nodes being used, this time to 5, so as to allow for shorter simulation periods.

Results

The purpose of this section is to document and discuss the results of each of the three experiments described in the previous section. The structure of this section is as follows; three main sub-sections are used to describe present the results and discuss their significance, for convenience, each sub-section begins with a brief overview of the experiment in order to avoid the need for constant re-reading of the detailed configurations presented in the previous section.

Experiment One – Efficiency and Reliability

To briefly recap, this experiment will use debug messages to print out both the total number of application packets sent by ordinary nodes and the number of application packets received by the base station. In the case of the un-modified Surge application, reliability will be calculated by subtracting the total received from the total sent and then expressing this value as a percentage. Efficiency will be taken to be a more subjective metric, largely based on the total number of packets sent, but also considering the amount received by the base station (an excessive number of packets would put an unnecessary strain on the base station.

For iHEED the determination of both reliability and efficiency are slightly more complicated than with Surge. Both of these were measured in two stages; node to cluster-head and cluster-head to base. Reliability is calculated in the same way as it was for Surge at both of these stages (i.e. packets received over packets sent) and the reliability of both stages are analysed separately before being combined to give an overall view of the network's reliability. As with Surge, efficiency will be looked at in terms of both packets being sent and packets received, however where packets sent will encompass both packets sent to cluster-heads as well as those sent directly to the base station, packets received by cluster-heads will not.

Appendix 8 contains a sample of the output from running the simulations, for brevity, only a small amount of the total output is included.

Surge

Analysis of the results using the Unix grep search facility (as described in the simulation configuration section) revealed that the base station received a total of 327 packets over the period of 1800 seconds (30 minutes). In comparison, a similar search for packets sent reveals that the 499 other nodes in the network sent a total of 13636 sensor packets over the duration of this experiment.

This means that out of the 13636 packets sent only 327 were received by the base station; therefore the reliability score for the Surge protocol is roughly 2.4%.

iHEED

With iHEED the total number of sensor data packets received by the base station was significantly lower with only 176 of these packets being received. This is almost 54% less than the number received by the base station running the un-modified Surge application. Although this may seem like a more efficient figure, it is important to remember that although iHEED uses data aggregation, the lower figure could be as a result of decreased reliability rather than increased efficiency, therefore it is important that reliability is assessed before any conclusions are drawn.

As discussed in the configuration section, iHEED will be examined using two types of reliability; cluster member to cluster-head and cluster-head to base. The first of these, yields somewhat surprising results; 47 packets were sent by cluster members and of those 22 were received by cluster-heads. What is most surprising about this figure is the low number of packets that have been sent by cluster members. Despite (or perhaps because of) the low number of packets, the reliability is a reasonable 47%.

An examination of the cluster-head to base station results reveals a similarly unexpectedly large number of cluster-head transmissions; 461, of which the base station received 176. This gives a reliability rating of around 38%. Therefore, by combining the two reliability stages an overall score of 42.5% is recorded.

Discussion

As the basic analysis shows, the results for the two applications are significantly different. In terms of sheer packets sent, Surge is clearly in the lead with exactly 13636 application packets being sent in comparison to the combined total of 508 packets sent by iHEED. Interpreted purely in terms of packets sent, iHEED is clearly the more efficient of the two, as despite being simulated with exactly the same number of nodes, and for exactly the same duration, its nodes have sent 13128 packets less than Surge (or 96%).

Although it was expected that iHEED would account for less traffic than Surge, the actual gulf between the two programs is still somewhat surprising. Although Younis and Fahmy (2005) state that iHEED contains mechanisms such as sending nodes on the outside of the network to sleep, as well as the obvious cluster-based data aggregation, that reduce the overall number of application packets being sent the gulf was still not expected to be this wide.

In light of this, it is probably not surprising that the base station running the iHEED application has received almost 54% fewer packets than the base station running the un-modified version of Surge. Although these results would probably indicate, that the iHEED application is far more efficient in its transmission of sensor readings than Surge, it is important to take the reliability of both applications into account before drawing such a conclusion.

With regards to the Surge application, it is fairly obvious from looking at the number of packets sent compared to the number received that it is somewhat un-reliable, and the reliability score of 2.4% reflects this. Obviously, due to the use of clustering and aggregation, it is slightly more complex with iHEED, however, from the results described previously it is clear that the inter-cluster score of 47% and the cluster-head to base node score of 38% both offer a significant improvement over Surge which is reflect in iHEED's overall rating of 42.5%.

As mentioned when the results were first given, although the inter-cluster score of 47% appears to indicate a reasonable level of reliability, the actual numbers of 47 packets sent and 22 received are seemingly incongruous. Figures such as these, when compared with the high number of packets sent by cluster-heads (461) would seem to indicate most nodes in the network are forming one-node clusters rather than electing to join a cluster led by another node.

Although this experiment did not record data related to the number of clusters formed, how long they remained active for, and how many nodes they contained, it is possible to make an informed guess at some of those questions by looking through the log (as shown in Appendix x). Of the 42 packets sent by cluster members, several of these appear to have been sent by the same node on several occasions, therefore, without actually counting the number of nodes that have sent them, it is estimated around 20 (possibly less) nodes were members of a cluster at some point during the simulation.

This means that (ignoring the base station) as few as 480 nodes acted as cluster-heads throughout the duration of the experiment. It is interesting to note that when measuring the performance of the original iHEED application, Younis and Fahmy (2005) do not give any information regarding the number of clusters formed, or their respective sizes. From an analysis of the given information as well as the iHEED source code, it is entirely possible that the clustering algorithm used could indeed lead to an abnormally large number of single-node clusters, and it is therefore recommended that future research into this area takes place.

In light of the unexpectedly high number of single-node clusters, it is not obviously apparent as to why the reliability of the network running the iHEED application is considerably superior to that of the Surge network. Although the original hypothesis that this experiment was based on predicted that iHEED would post a significant improvement in terms of efficiency and reliability, it was expected that this would be as a result of clustering and data aggregation. Clearly, the improved reliability and efficiency cannot be primarily attributed to either of those mechanisms, which then begs the question; if not clustering and aggregation, then what is responsible for the improved results?

A review of both the original paper as well as the iHEED documentation presents very few clues, one of which is the use of the sleep mechanisms on selected nodes (Younis and Fahmy, 2005). As discussed earlier, in order to save energy, iHEED sends nodes that are classed as ‘lower-tier’ nodes to sleep for significant durations (Younis and Fahmy, 2005). What this means is that nodes who are not cluster-heads use a ‘sleep-to-work’ ration in which they are allowed to enter sleep mode once they have completed their task (Younis and Fahmy, 2005). This is in contrast to the original Surge application which contains no such mechanism, nor does it attempt to in any way alter or set the rate at which nodes transmit sensor readings.

Therefore, the use of sleep-to-work rations in iHEED could be seen as a contributing factor to the improvement in transmission efficiency. There is, of course, one slight problem with this theory; only ordinary nodes (i.e. nodes which are not cluster-heads) are allowed to take advantage of this sleep-to-work ratio. Considering the large number of nodes which acted as cluster-heads, it would seem that this would be the nail in the coffin of that theory, however, it is important to remember that clustering is not infinite, and a node that was previously an ordinary node could wake-up and subsequently become a cluster-head (remembering that cluster-head election is predominantly based on residual energy, which a node that was allowed to rest would presumably have high levels of).

Again, without further analysis of the clustering process it is impossible to determine whether or not this could have been a contributing factor to the increased efficiency displayed by the iHEED application. Moving away from efficiency, it is even less clear why iHEED has a substantially improved level of reliability than Surge. Taking efficiency into account, it is possible that the significantly reduced congestion has led to an increase in successful transmissions, albeit not as a result of data aggregation.

Finally, regarding the hypothesis that this experiment was designed to answer (clustering and data aggregation will ensure more efficient transmission of sensor readings without negatively effecting reliability) the results are somewhat ambiguous as to whether the hypothesis was correct. Although the end goals of increased efficiency and reliability have undoubtedly been met, as the attempts to determine to what factor this is due attest, data aggregation and clustering have only had a limited effect on the efficiency and reliability of the iHEED network.

Therefore, to answer the hypothesis; the iHEED application does exhibit improved efficiency and reliability, however, this is most likely not as a result of clustering and data aggregation with the exact cause remaining unknown, pending future investigation.

Experiment Two – Energy Consumption

As was discussed in the last section, like Experiment One, this experiment captures several metrics (all of which are related to energy consumption) in the one simulation. However, unlike Experiment One the setup for this experiment is far simpler; the simulation is configured to collect power debug messages and run the PowerTOSSIM plugin (described in detail earlier) to collect detailed information regarding each node's use of power and energy.

The results of this simulation are then analysed by the PostProcess tool in order to obtain a breakdown of the total energy used by each component (i.e. radio, CPU, sensing device, etc.) as well as the overall energy consumption of each node (Schnayder, 2004). From this analysis several metrics (or perhaps more accurately sub-metrics) can then be obtained; overall energy consumption, average energy consumption, cost of communication (the total energy consumed by the radio component), etc.

Appendix 9 contains a sample of the output from running the PostProcess, for brevity, only a small amount of the total output is included.

Surge

Before looking at the totals and averages, it is important to examine the actual analysis, shown in Appendix x, there are two components that seemingly consume no energy at all; the first is ADC and the second is LEDs. In the case of LEDs this is not surprising, although iHEED makes frequent use of LEDs to indicate its status, Surge does not do so at all, and the only line of code which makes reference to the LEDs is, see Appendix x for more detail.

As for the missing ADC measurements this isn't as obvious, but there is still a reason. The reason is found in the original paper describing the operation of PowerTOSSIM (Schnayder, 2005) which states that from the readings of actual nodes taken, the ADC component did not seem to consume a significant amount of energy with any of the nodes, and therefore the Post Process tool does not count ADV operation (although PowerTOSSIM still records ADC operation). Although the LED absence only affected Surge, as the same power model was used for the analysis of iHEED, there will be no ADC measurements for it either.

An important aspect of energy consumption is transmission cost, as discussed at numerous points in this report, the radio is typically the biggest consumer of energy, and an important metric for this project is the amount of energy consumed by the radio relative to the total energy consumed. For Surge, the total amount of energy consumed by all nodes in the network is 1271932.96 mJ (1.27 kJ) while the average energy consumed by a node running the Surge application is 12719.3296 mJ (12.72 Joules).

Although the cost of transmission in terms of energy used by the radio component is often the biggest consumer of energy, it is important that the total energy used by the application is looked at. Of the 100 nodes simulated, the total energy spent by all nodes running the Surge application was 2139076.8 mJ (2.14 kJ) while the average energy spent by a node was 21390.768 mJ (21.39 Joules).

This means that of the combined energy measurements, approximately 59.3% (1.27 kJ) of all energy spent was used by the radio, while the average for a node running Surge was 69.4% (12.72 Joules).

iHEED

As with the original Surge application there are no ADC measurements, however, there are measurements for LEDs which are made use of frequently, as can be seen from the code in Appendix x.

With this in mind, the total energy consumed by all nodes running the iHEED application is 2370193.25 mJ (2.4 kJ) while the average energy consumed by a node running iHEED was 23701.9325 mJ (23.7 Joules).

Regarding the cost of transmission, the total amount of energy consumed by the radio components of all nodes is 1253111.44 mJ (1.25 kJ), while the average for a node running the iHEED application is 12531.1144 mJ (12.53 Joules).

When compared with the total energy consumption for all nodes, the radio component was responsible for approximately just under 48% (1.15 kJ) of all energy consumed, while the average cost of transmission is 47.13% (11.7 Joules) of the overall energy consumed by a node.

Discussion

As with the first experiment, the results are again surprising, as although the iHEED application did indeed decrease the cost of transmission by a total of 22% which meant an average reduction

of around 11% per node. Although this was indeed expected, and mentioned in the hypothesis, what was not expected was that the overall energy consumed by nodes running iHEED would be higher in terms of both total (around 200 Joules) and average (2.3 Joules). The most logical reason for increased energy consumption in spite of decreased transmission cost is most likely due to iHEED's frequent use of LEDs, without performing a detailed analysis viewing the Post Process results for iHEED shows that in some nodes the energy consumed by LEDs is as high as 5 Joules!

Although one of the main differences between Surge and iHEED is that iHEED is designed for real-world use on actual nodes whereas Surge is a demonstration application, therefore it is perhaps more desirable that iHEED makes use of LEDs to indicate its status to human observers. However, the fact that despite reducing the cost of transmission the overall energy consumption is still higher than Surge's so this would perhaps suggest that LEDs were being used inefficiently, and their role in the iHEED application should be revised.

Moving away from overall energy consumption and looking at the statistics for cost of transmission, it is interesting to note the difference (in terms of percentage) between the total (59.3%) and average (69.4%) for the Surge application which is around 11.1%. By contrast the difference between the total (48%) and average (47.13) cost of transmission is less than 1% for the iHEED application. What this would seem to suggest is that not only does iHEED offer a reduced cost of transmission, but that due to the almost negligible difference between total and average, it also offers a better distribution of radio use. On the other hand, Surge has a higher cost of transmission, and as the significant difference between total and average cost of transmission would seem to indicate an inconsistent pattern.

This is not really surprising, looking back at the previous experiment iHEED was found to have sent 96% less application packets than Surge. Therefore it comes as no surprise that despite the odd results relating to clustering discussed in the previous experiment that the amount of energy spent on sending and receiving messages has been reduced. What is odd is the difference of around 11% in the total and average transmission costs for the Surge application. As the Surge application has nodes sensing and sending at a constant rate this is somewhat surprising as it would be expected that most nodes would have a similar transmission cost.

Perhaps a possible explanation for this could be the routing system used by Surge, maybe what skewed the results was node's location so that nodes near the base station received more traffic,

or nodes further away had to use greater energy to reach the base. Although iHEED uses a variant of Surge's multi-hop routing, in addition to implementing clustering and aggregation, it also alters the way in which routing trees are formed and uses energy and link reliability to determine which node to send data to (Younis and Fahmy, 2005). Therefore by being able to determine how much energy it would cost to send a packet to a particular node would obviously allow nodes to choose more energy efficient paths. Similarly, by being able to asses link availability, nodes would to pick routes that were more reliable and thus reduce the need for retransmission

Similarly, other features such as cluster-head rotation and allowing non-critical nodes to sleep for set periods of time might also have helped keep the cost of transmission lower as well as ensuring that the distribution of transmission costs is also more equal. Although given the results of the first experiment seemed to prove that iHEED's clustering was somewhat flawed and ineffective the validity of cluster-head rotation is doubtful.

Considering the hypothesis, as with the previous experiment the results are again ambiguous in the sense that given the results of the first experiment it appears clustering and aggregation didn't play as big a role as expected. However, with this experiment there is the added ambiguity of iHEED having higher overall energy consumption despite decreased cost of transmission. Therefore, to answer the hypothesis although iHEED did reduce the cost of transmission, it also consumed more energy overall than Surge.

Experiment Three – Network Lifetime

When this experiment was conceived, it was envisaged that both applications would be allowed to run until the base station ceased to work, the point at which the base station died would then be deemed to be the lifetime of the network. Unfortunately, when it was time to perform this final experiment, it was deemed that there was not enough time to leave the two applications running until the death of the base station (in initial tests with the Surge application the base station was still functioning after 11 simulation days) as the other experiments had been performed first.

Therefore, a revision of the experiment described earlier was performed in which both applications would be simulated for 72 hours of simulation time, and when the simulation was complete the number of nodes alive would be taken. As no nodes died in either simulation, this

experiment has technically failed in its aims as it has only really proven that the lifetime of networks running both applications is at least 72 hours (3 days) and no doubt significantly more.

Therefore, in light of the limited time and resources available, this experiment is inconclusive and it is recommended that further testing, in which simulations of both applications are allowed to run until the base station's death is performed.

Conclusion

Project Aims and Objectives

Throughout this report it has been stated that the overall goal of this project has been to assess the effects that clustering and data aggregation have on Quality of Service in wireless sensor networks. More specifically, this was to involve the simulation of the iHEED application running in the TinyOS environment and using the TOSSIM simulator. The reason for the choice of the iHEED application was that it combined the HEED clustering algorithm with data aggregation and multi-hop-routing in a TinyOS application (Younis and Fahmy, 2005).

From these general aims, the following research question was developed:

“How effective is the iHEED application’s combination of clustering and data aggregation at improving Quality of Service in a TinyOS-based Wireless Sensor Network?”

This research question was then used as the basis for all other planning, the direct product of which was the project's overall objectives, and from these more detailed literature review and primary research objectives were derived. In order to help determine how successful the project has been in meeting it's overall aim it is important not just to look at the results, but to also look at the main objectives and determine how successfully each of these were fulfilled.

Objective One: Research Wireless Sensor Networking

Although it may sound like one of the simpler objectives, this was actually one of the most important stages of the project, as without completing it successfully, it is unlikely any of the other objectives could have been achieved. This objective largely took the form of a literature review and involved, reading up on numerous topics related to wireless sensor devices, applications, and networking.

The main product of this objective was a portion of the introduction and literature review section of this report dedicated to background information. However, the total number of papers, books,

and websites that were read in the process of completing this stage is well over 100, and only a few of them have made it into this report. Even those that were not included in here were mostly of use, even if was something as small as helping to find other, more relevant, sources of information.

Therefore, due to the significant amount of reading undertaken, and the subsequent use of the knowledge acquired in doing so, both in writing this report and assisting in achieving other aims and objectives, this objective can be judge to have been successfully fulfilled.

Objective Two: Understand WSN QoS

Like the first objective, the most obvious product of this objective would be a portion of the literature review. However, as with the first objective, its long-term significance has proven to be greater than producing the literature review.

The purpose of this objective was not to acquire knowledge for the sake of it, but to do so in order to gain an understanding of QoS, first in general terms, and then with relevance to wireless sensor networks. The reason that doing so is important should be obvious; the aim of this project is to investigate wireless sensor network QoS! Therefore it was vital that an understanding of WSN QoS was obtained well before the primary research stage commenced.

Aside from the literature review, another important product of this stage was the definition of QoS. It was eventually concluded that in the area of wireless sensor networking there is almost certainly no such thing as an unequivocal, universal definition of QoS, and that the level of QoS (and the appropriate metrics) are somewhat dependent on the application. This meant, that when the iHEED application was looked at later on, that suitable metrics for evaluating the level of QoS it provided could be derived using the knowledge acquired through this objective. Therefore, this objective could be judged to have been successfully fulfilled.

Objective Three: Research Clustering and Data Aggregation Techniques

Again this is another objective where the main product of it is the literature review. However, this objective represents an increasingly narrow focus, as the project is now beginning to look at the specific technologies it is investigating and acquire knowledge about these. This knowledge

can then be used to examine the more specific HEED clustering protocol in comparison with other approaches. Similarly, given the somewhat vague information about iHEED's data aggregation protocol, the general knowledge of what data aggregation is, why it's used, and how it works that was acquired at this stage was undoubtedly useful later on.

By acquiring knowledge here that would prove to be useful in both the examination of a particular clustering protocol (HEED) as well as being useful in evaluating the operation of the iHEED application, this objective has been successfully completed.

Objective Four: Research HEED and iHEED

Where the previous objective represented a narrowing of the project's focus, this objective could be said to have acted as a bridge between the literature review and primary research stages of the project. By this stage it had already been established that iHEED would be the focus of the practical stage, therefore a basic knowledge of the application has obviously been acquired, however, it was just that – basic!

The purpose of this objective was to not merely read academic papers, but also explore the source code and the environment (TinyOS) although the actual implementation of the iHEED application did not take place at this stage, TinyOS had been installed and several other applications had been simulated in order to gain insight into how the system worked. In order to better understand how iHEED worked, several manuals and technical sources were reviewed so as to gain a rudimentary understanding of the nesC language that iHEED and TinyOS is programmed with.

The practical knowledge gained at this stage was absolutely vital in helping to achieve later objectives, most notably the implementation stage where iHEED was adapted to run in TOSSIM, and then subjected to further modifications to collect data. Similarly, the more theoretical knowledge was directly applicable to the next objective (design the testing and evaluation scheme) as well as latter stages which required analysis of results. With this in mind, it would be fair to say that this objective was successfully completed.

Objective Five: Design the Testing and Evaluation Scheme

Building on the knowledge acquired through the previous objectives, the primary purpose of this objective was to make use of this knowledge by devising a test scenario, metrics, and a strategy. Naturally this involves elements of various previous objectives (i.e. the knowledge of QoS acquired in the second objective) in order to not only come up with a testing scheme that was practical, but that represented a fair and accurate measurement of QoS.

This required a knowledge of not only metrics, but of the particular features of wireless sensor networks in general and TinyOS nodes running the iHEED application in particular. Similarly, in order to determine the viability of particular tests it was necessary to have obtained some practical experience of the TinyOS environment.

By combining both practical and theoretical knowledge, metrics (or groups of metrics) were devised and from these a testing plan and implementation strategy were drawn up. Although the results were somewhat ambiguous and the lifetime experiment poorly implemented, this was not as a result of any flaws in the testing plan, but in the actual implementation. Therefore, despite the flaws of the implementation, this objective could be considered successfully completed.

Objective Six: Implement and Test the iHEED Application

Up until this stage, it has been clear that all of the previous objectives seem to have been completed without any problems, and it would be true to say that up until this stage the project was running smoothly. Several initial problems with the iHEED application, as well as the lack of readily available information (there are no text books, and only one TinyOS manual) led to the need for more time being devoted to debugging iHEED than was originally planned.

The upside of this is that iHEED was eventually modified so as to run on the TOSSIM simulator, for possibly the first time. Although there are no academic papers detailing successful implementation of iHEED in TOSSIM, it is possible that it has been done. However, after checking both academic sources as well as the TinyOS newsgroup, it is unlikely any implementation has been publicised. Therefore, by successfully modifying iHEED to run in TOSSIM this project has made a definite contribution to TinyOS and wireless sensor research.

As the implementation stage had taken longer than expected this also affected the testing stage, which had several problems of its own – namely in terms of overcoming the complexities of

measuring reliability in a system that used clustering and data aggregation – and these only delayed the project further. An unfortunate side-effect of these delays was that work relating to the network lifetime was pushed further and further back, which meant that when it was eventually simulated time and resources were extremely constrained meaning a full evaluation was impossible. This is, in hindsight, a complete error of planning that unfortunately had a serious effect on the quality of the results.

Unlike the previous objectives, which only had slight problems, this was the stage that had the most problems, however, the modification of iHEED was a major success, therefore, despite the problems encountered, and the ambiguity of results, it would be hard to deem this objective a complete failure. Therefore, it is perhaps best to think of it as a partially completed objective.

Objective Seven: Evaluate the Test Results and Draw Conclusions on the Effectiveness of iHEED:

Because of the problems encountered during the previous objective, the subsequent evaluation of the test results was inconclusive. However, it is important to acknowledge that the failings of the previous objective should not decide the success of this objective. Even if the results were not as expected, that does not mean that a reasonable effort to analyse and extrapolate some sort of conclusion from them cannot be made. The analysis of the results is not a simple question of looking at the relevant hypothesis and determining whether it has been fulfilled (although this is an important part of the analysis process) but it also involves trying to explain results – especially anomalous results – using knowledge of iHEED (or Surge), TinyOS, and sensor networks in general.

Looking through the list of objectives, it seems as if most of these have been at least partially fulfilled, however, it should be noted that neither the project question or the hypotheses have actually been answered. After all, the whole purpose of the project was to answer that question, and rather than push this somewhat significant aspect into the same sub-section as the objectives, the next sub-section will look at whether or not the project question and hypotheses have been answered.

Research Question and Hypotheses

In addition to the project question, three hypotheses were developed, and from these three experiments were derived in order to investigate the hypotheses (although, naturally there is some overlap between the three). The purposes of these hypotheses were to split the project question up into stages, so to speak, and at each stage a particular aspect of QoS (i.e. reliability) was examined in order to determine what effect the modifications to iHEED had on that metric.

Hypotheses

In order to determine whether the overall research question has been answered, each of these hypotheses will be revisited and the performance of iHEED (in contrast to Surge) evaluated.

Hypothesis One: clustering and data aggregation will ensure more efficient transmission of sensor readings without negatively effecting reliability

On an initial inspection of the results, it would seem as if iHEED was indeed both more reliable and more efficient than Surge, however, although these two claims are technically, as was discussed earlier a rather odd statistic is that of the 508 packets sent by nodes running iHEED, only 47 packets were sent by cluster members, while the rest were sent by cluster-heads. Although it is impossible to say for sure, this would seem to indicate that a large number of one-node clusters were being formed, which would indicate a deficiency in the clustering process. As iHEED was only tested on a small number of nodes by its authors (Younis and Fahmy, 2005) and has not been subject to independent testing it is possible that the clustering protocol does not scale to large numbers.

Another possible reason is that the application had been implemented incorrectly and thus the results are also incorrect. However, this is unlikely as before formal simulation extensive testing (as detailed in the methodology) was performed which seemed to verify that iHEED was acting as it should. A more logical possibility is that although iHEED's clustering produced odd results, in addition to clustering iHEED featured other improvements such as routing decisions based on

energy and link reliability (Younis and Fahmy, 2005) and they could be responsible for the improved reliability.

To answer the hypotheses, however, it is indeed true that iHEED achieved a higher level of efficiency and reliability, however, it is unknown exactly what the cause of this was, although it is unlikely that clustering and aggregation played a major role.

Hypothesis Two: through data aggregation and clustering, the average cost of sending and receiving a message (as expressed in energy consumed by the radio) will be reduced as will the average overall energy consumption

The results for this hypothesis's experiment was again somewhat surprising although not quite as surprising as with the previous one! The main surprise here was that although iHEED did indeed have a significantly lower total and average cost of transmission (48% and 47.13% respectively) than Surge (59.3% and 69.4% respectively) its overall energy consumption both in terms of total and average (2.1 kJ and 23.7 Joules respectively) was higher than Surge's (2.14 kJ and 12.72 Joules respectively). Given that in both cases Surge's actual amounts of energy spent by the radio component was slightly higher than iHEED's, the results seemed somewhat odd.

It is however, important to bear in mind that iHEED makes frequent use of LEDs to signal its status, whereas Surge doesn't. Although conclusive testing was not performed, it would appear as if this was responsible for the increased energy consumption. If this is found to be true, then perhaps the use of LEDs in iHEED should be revised, as although they are useful for providing human observer's with an idea of what is going on, it is surely not worth wasting limited power?

Again, it is difficult to answer the hypothesis directly, due to the knowledge that the clustering portion of iHEED produced strange results in the previous experiment, but also due to iHEED having a lower cost of transmission, yet higher overall energy consumption. Therefore, although iHEED does indeed lead to a reduced cost of transmission, its frequent use of LEDs (and possibly other factors) means that its overall energy consumption is worse than Surge's.

Hypothesis Three: despite the increased strain on cluster-heads, the reduction in traffic, and periodic re-clustering will not significantly decrease overall network lifetime

As the results section has explained, due to poor planning and time management, the testing of this hypothesis was somewhat inadequate, and succeeded only in proving that both application's had a lifetime of at least 3 days.

Project Question

“How effective is the iHEED application’s combination of clustering and data aggregation at improving Quality of Service in a TinyOS-based Wireless Sensor Network?”

As the analysis of the results in relation to the hypotheses above show, the results are somewhat mixed. Although iHEED posted improved efficiency, reliability, and reduced cost of transmission, due to the low number of messages sent by cluster members it is unclear what part clustering and data aggregation played in these results. Another anomaly is iHEED having a higher overall energy consumption than Surge, although this is possibly as a result of its use of LEDs.

Regarding network lifetime, as has been discussed in the previous sub-section, as well as the results section, due to unfortunate constraints it is unclear exactly what effect iHEED's modifications have had on its lifetime, other than both it and Surge are able to run for at least three days.

Therefore, to answer the research question, it is unclear exactly how effective clustering and data aggregation have been with regards to WSN QoS. Although it is clear that in most areas (reliability, efficiency, cost of transmission) iHEED has lead to an improved level of QoS. Therefore, the recommendation of this project is that in light of these results further, more detailed investigation of the iHEED application is performed.

Project Critique

It's been mentioned several times that various problems were encountered throughout the lifetime of this project, however, most of them were largely resolved by the simulation stage, although due to the time spent working on them, the lifetime simulation had to be modified, and was most likely an inadequate measurement.

Although some of these problems were somewhat unavoidable, due to the decision to use an application that was not designed for simulation, and then adapt it, the fact remains that the details of how network lifetime would be calculated were decided as far back as February. Therefore, as soon as iHEED was confirmed to be fully operational simulation of network lifetime could have begun

It would seem obvious, then, that network lifetime should have been simulated as early on as possible so as to get it out of the way and allow for work on determining how other metrics should be collected without impacting on network lifetime measurements. However, at the time, determining how the other metrics should be collected was deemed a more important task, and so network lifetime was left until the last minute, which naturally had a negative effect on the accuracy of its results.

In addition to this, if time had been used more efficiently, then it might have been possible to conduct further experiments that would have allowed for a more detailed analysis as to why so many one-node clusters were formed. Similarly, the process of analysing the clustering and data aggregation aspects of iHEED could have perhaps been expanded. Indeed, it was originally intended that these aspects would be looked at in detail, however this idea was dropped as it would only apply to iHEED and would not allow for a direct comparison with Surge.

Despite these setbacks and shortcomings the project can also have been considered to have achieved some significant successes. As stated earlier, this is the first (known) independent academic analysis of the iHEED protocol, and although the results are inconclusive, this project can still serve as the basis for future results, both in terms of the new issues posed by the results, as well as by attempting to answer the same questions through slightly different methodology. Another major contribution that this project has made is the simulation of the iHEED application for the first time. Although the actual modifications only accounted for a few lines of code, determining the modifications as well as testing the application to ensure it was performing correctly took significant time and effort.

Therefore, by modifying and simulating the iHEED application for the first time, despite the surprising results and the limited time and resources available, this project has still managed to make a contribution to the area of wireless sensor network QoS.

Future Work

Perhaps the most obvious area of future work is further examination of the clustering process used by iHEED. Due to the surprising number of one-node clusters, it is important that any future work attempts to address this issue by performing more research into how clustering in iHEED works. Possible areas to look at would include; how many nodes become cluster-heads, how many members their clusters has, how many times a node becomes a cluster-head, and how many nodes never join a cluster.

As well as looking at these aspects, future work could attempt to change the number of nodes and length of time being simulated, so as to determine how scalable iHEED is. Similarly, another area for future research would be energy consumption. A more rigorous analysis of energy consumption could lead to the identification of the reason why iHEED has a lower communication cost than Surge, but higher overall energy consumption.

Following such an analysis modifications could be made to iHEED to see if greater energy savings could be made. Other possible modifications could include an improved clustering mechanism, pending the results of the investigation into iHEED's clustering or maybe even try a different data aggregation model.

One important area of future work that should not be neglected is network lifetime. It has been stated frequently that because it was left to the last minute, simulation of this was inadequate, and therefore the results were inconclusive. Any future research should make it a priority to allocate time and resources to calculating network lifetime due to its obvious importance in sensor network applications.

Conclusions

Despite the numerous setbacks and problems encountered, some of which were as a result of poor planning and bad time management, while others were completely unforeseen and thus would have been hard to avoid, this project has managed to produce a contribution to the field of wireless sensor networking.

This contribution is the simulation of the iHEED application, although the results were surprising to say the least, and the investigation process could have perhaps been more comprehensive this project undeniably chartered new waters. Therefore, even if the results themselves prove to be of little real-world use, the project itself can still serve as the basis for future work.

Speaking of which, with regards to the project question, although it is unclear what sort of role clustering and data aggregation played, the iHEED application does offer an improvement over Surge in terms of Quality of Service. Exactly how is not known, but should be clarified by future work – using this project as a starting point.

Reference List

Abbasi, A.A. & Younis, M. 2007, "A survey on clustering algorithms for wireless sensor networks", *Comput.Commun.,* vol. 30, no. 14-15, pp. 2826-2841.

Adiseshu, H., Parulkar, G. & Yavatkar, R. 1998, "A state management protocol for IntServ, DiffServ and label switching", *Network Protocols, 1998. Proceedings. Sixth International Conference on*, pp. 272.

Akyildiz, I.F., Su, W., Sankarasubramaniam, Y. & Cayirci, E. 2002, "Wireless sensor networks: a survey", *Computer Networks,* vol. 38, no. 4, pp. 393-422.

Barbancho, J., Leon, C., Molina, J. & Barbancho, A. 2006, "Giving neurons to sensors. QoS management in wireless sensors networks.", *Emerging Technologies and Factory Automation, 2006. ETFA '06. IEEE Conference on*IEEE, , pp. 594.

Biyu, L., Frolik, J. & Wang, X.S. 2005, "A predictive QoS control strategy for wireless sensor networks", *Mobile Adhoc and Sensor Systems Conference, 2005. IEEE International Conference on*, pp. 8 pp.

Çam, H., Özdemir, S., Nair, P., Muthuavinashiappan, D. & Ozgur Sanli, H. 2006, "Energy-efficient secure pattern based data aggregation for wireless sensor networks", *Computer Communications,* vol. 29, no. 4, pp. 446-455.

Chamam, A. & Pierre, S. 2009}, "A Distributed Energy-Efficient Cluster Formation Protocol for Wireless Sensor Networks", *2009 6TH IEEE CONSUMER COMMUNICATIONS AND NETWORKING CONFERENCE, VOLS 1 AND 2}*, pp. 504.

Chen, D. & Varshney, P.K. 2004, "QoS Support in Wireless Sensor Networks: A Survey", *International Conference on Wireless Networks*, pp. 227.

Choe, H.J., Ghosh, P. & Das, S.K. 2009, "Cross-Layer Design for Adaptive Data Reporting in Wireless Sensor Networks", *2009 Ieee International Conference on Pervasive Computing and Communications (Percom), Vols 1 and 2; 7th IEEE International Conference on Pervasive Computing and Communications*IEEE, , pp. 525.

Choi, H., Wang, J. & Hughes, E. 2009, "Scheduling for information gathering on sensor network", *Wireless Networks,* vol. 15, no. 1, pp. 127.

Cisco 2004, "Quality of Service (QoS)" in *Internetworking Technology Handbook*, Fourth edn, Cisco Press, Indianapolis, pp. 49-1-49-32.

Dai, J. & Liu, S. 2009}, "A Novel Clustering Algorithm in Wireless Sensor Networks", *2009 INTERNATIONAL CONFERENCE ON INDUSTRIAL AND INFORMATION SYSTEMS, PROCEEDINGS}*, eds. Q. Luo & X. Tian, , pp. 159.

de Freitas, E.P., Wehrmeister, M.A., Pereira, C.E. & Larsson, T. 2008, "Real-time support in adaptable middleware for heterogeneous sensor networks", *Computer Science and Information Technology, 2008. IMCSIT 2008. International Multiconference on*, pp. 593.

Deng Shuguang, Shen Lianfeng, Zhu Xiaorong & Lin Lin 2008, "A Novel On-demand Qos Protocol Based on Near Static Clustering for Wireless Sensor Network", *Isise 2008: International*

Symposium on Information Science and Engineering, Vol 2; International Symposium on Information Science and Engineering, eds. F. Yu, G. Yue & Z. Chen, , pp. 571.

Dulman, S., Chatterjea, S. & Havinga, P. 2006, "Introduction to Wireless Sensor Networks" in *Embedded Systems Handbook*, ed. R. Zurawski, 1st edn, CRC Press, Boca Raton, Florida, pp. 31-1-31-10.

El-Gendy, M.A., Bose, A. & Shin, K.G. 2003, "Evolution of the Internet QoS and support for soft real-time applications", *Proceedings of the IEEE,* vol. 91, no. 7, pp. 1086-1104.

Enel, L. & Martinet, L. 2005, "Application of QoS management techniques to new naval combat systems", *Oceans 2005 - Europe*, pp. 1238.

Estrin, D., Girod, L., Pottie, G. & Srivastava, M. 2001, "Instrumenting the world with wireless sensor networks", *Acoustics, Speech, and Signal Processing, 2001. Proceedings. (ICASSP '01). 2001 IEEE International Conference on*, pp. 2033.

Frye, L.M. 2007, "Wireless sensor networks: learning and teaching", *SIGITE '07: Proceedings of the 8th ACM SIGITE conference on Information technology education*ACM, New York, NY, USA, pp. 269.

Gay, D., Levis, P., Behren, R.v., Welsh, M., Brewer, E. & Culler, D. 2003, "The nesC language", *ACM SIGPLAN Notices,* vol. 38, no. 5, pp. 1-11.

Gelenbe, E. & Ngai, E.C.-. 2008, "Adaptive QoS routing for significant events in wireless sensor networks", *Mobile Ad Hoc and Sensor Systems, 2008. MASS 2008. 5th IEEE International Conference on*, pp. 410.

Goldsmith, D.L., Liebowitz, B., Park, K., Wang, S., Doshi, B. & Kantonides, J. 2006, "Precedence and Quality of Service (QoS) Handling in IP Packet Networks", *Military Communications Conference, 2006. MILCOM 2006. IEEE*, pp. 1.

Gowrishankar, S., Basavaraju, T.G., Manjaiah, D.H. & Sarkar, S.K. 2008, "Issues in Wireless Sensor Networks", *WCE 2008: World Congress on Engineering 2008,* , pp. 176-187.

Guo, B. & Li, Z. 2009, "A dynamic-clustering reactive routing algorithm for wireless sensor networks", *Wireless Networks,* vol. 15, no. 4, pp. 423.

Hamdy, M. & El-Madbouly, H. 2007, "Improvement of QOS Management in Wireless Sensor/Actuator Networks Using Fuzzy-Genetic Approach", *Icnm: 2009 International Conference on Networking & Media Convergence; International Conference on Networking and Media Convergence*, eds. H. Fahmy, A. Salem, M. ElKharashi, A. BahaaElDin & M. Taher, , pp. 29.

Heinzelman, W.R., Chandrakasan, A. & Balakrishnan, H. 2000, "Energy-Efficient Communication Protocol for Wireless Microsensor Networks", *HICSS '00: Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8*IEEE Computer Society, Washington, DC, USA, pp. 8020.

Hill, J., Horton, M., Kling, R. & Krishnamurthy, L. 2004, "The platforms enabling wireless sensor networks", *Commun.ACM,* vol. 47, no. 6, pp. 41-46.

Hoes, R., Basten, T., Tham, C., Geilen, M. & Corporaal, H. 2009, "Quality-of-service trade-off analysis for wireless sensor networks", *Performance Evaluation,* vol. 66, no. 3-5, pp. 191-208.

Howitt, I., Manges, W.W., Kuruganti, P.T., Allgood, G., Gutierrez, J.A. & Conrad, J.M. 2006, "Wireless industrial sensor networks: Framework for QoS assessment and QoS management", *ISA transactions,* vol. 45, no. 3, pp. 347-359.

Karl, M. 2005, "A comparison of the architecture of network simulators ns-2 and tossim", *Institut fr Parallele und Verteilte Systeme, Abteilung Verteilte Systeme*Universität Stuttgart, .

Kartvelishvili, M. 2003, "IP Qos Architectures", *NATA Advanced Technology Workshop*.

Kothari, N., Millstein, T. & Govindan, R. 2008, "Deriving State Machines from TinyOS Programs Using Symbolic Execution", *IPSN '08: Proceedings of the 7th international conference on Information processing in sensor networks*IEEE Computer Society, Washington, DC, USA, pp. 271.

Krishnamachari, B., Estrin, D. & Wicker, S.B. 2002, "The Impact of Data Aggregation in Wireless Sensor Networks", *ICDCSW '02: Proceedings of the 22nd International Conference on Distributed Computing Systems*IEEE Computer Society, Washington, DC, USA, pp. 575.

Langendoen, K. & Reijers, N. 2003, "Distributed localization in wireless sensor networks: a quantitative comparison", *Computer Networks,* vol. 43, no. 4, pp. 499-518.

Levis, P., Lee, N., Welsh, M. & Culler, D. 2003, "TOSSIM: accurate and scalable simulation of entire TinyOS applications", *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*ACM, New York, NY, USA, pp. 126.

Lu, Y. & Sheu, T. 2007, "An efficient routing scheme with optimal power control in wireless multi-hop sensor networks", *Computer Communications,* vol. 30, no. 14-15, pp. 2735-2743.

Malesci, U. & Madden, S. 2006, "A Measurement-Based Analysis of the Interaction Between Network Layers in TinyOS" in *Lecture Notes in Computer Science volume 2868/2006* Springer, Berlin, pp. 292-309.

Manjeshwar, A.D.P.A. 2001, "TEEN: A Routing Protocol for Enhanced Efficiency in Wireless Sensor Networks", , ed. Dharma P. Agrawal, , pp. 30189a.

Merrett, G.V., White, N.M., Harris, N.R. & Al-Hashimi, B.M. 2009, "Energy-Aware Simulation for Wireless Sensor Networks", *Sensor, Mesh and Ad Hoc Communications and Networks, 2009. SECON '09. 6th Annual IEEE Communications Society Conference on*IEEE, .

Mhatre, V. & Rosenberg, C. 2004, "Design guidelines for wireless sensor networks: communication, clustering and aggregation", *Ad Hoc Networks,* vol. 2, no. 1, pp. 45-63.

Min, R., Bhardwaj, M., Seong-Hwan Cho, Shih, E., Sinha, A., Wang, A. & Chandrakasan, A. 2001, "Low-power wireless sensor networks", *VLSI Design, 2001. Fourteenth International Conference on*, pp. 205.

Mohapatra, P., Li, J. & Gui, C. 2002, "Qos in Mobile Ad Hoc Networks", *Special Issue on QoS in Next-Generation Wireless Multimedia Communication Systems in IEEE Wireless Communications,* vol. 10, no. 3, pp. 44-57.

Monowar, M.M., Rahman, M.O., Choi, B.G. & Hong, C.S. 2008, "A Hop by Hop Rate Control Based QoS Management for Real Time Traffic in Wireless Sensor Networks", *Challenges for Next Generation Network Operations and Service Management, Proceedings; LECTURE NOTES IN COMPUTER SCIENCE; 11th Asia-Pacific Network Operations and Management Symposium*, eds. Y. Ma, D. Choi & S. Ata, , pp. 177.

Munir, S.A., Bin, Y.W., Biao, R. & Jian, M. 2007, "Fuzzy logic based congestion estimation for QoS in wireless sensor network", *2007 Ieee Wireless Communications & Networking Conference, Vols 1-9; IEEE Wireless Communications and Networking Conference; IEEE Wireless Communications and Networking Conference*IEEE, , pp. 4339.

Peplnjak, I. 2008, 10/10-last update*, The history and future of QoS* [Homepage of TechTarget ANZ], [Online]. Available: http://searchvoip.techtarget.com.au/articles/27317-The-history-and-future-of-IP-QoS [2010, 02/06] .

Perillo, M.A. & Heinzelman, W. 2005, "Wireless Sensor Network Protocols" in *Algorithms and Protocols for Wireless and Mobile Networks*, ed. A. Boukerche, 1st edn, CRC Hall Publishers, .

Peterson, N., Anusuya-Rangappa, L., Shirazi, B.A., Renjie Huang, Wen-Zhan Song, Miceli, M., McBride, D., Hurson, A. & LaHusen, R. 2009, *TinyOS-Based Quality of Service Management in Wireless Sensor Networks*.

Rahman, M.O., Monowar, M.M. & Hong, C.S. 2008, "A QoS adaptive congestion control in wireless sensor network", *10th International Conference on Advanced Communication Technology, Vols I-Iii - Innovations Toward Future Networks and Services; 10th International Conference on Advanced Communication Technology*ETRI, , pp. 941.

Romer, K. & Mattern, F. 2004, "The design space of wireless sensor networks", *IEEE Wireless Communications Magazine,* vol. 11, no. 6, pp. 54-61.

Sammapun, U., Lee, I., Sokolsky, O. & Regehr, J. 2007, "Statistical Runtime Checking of Probabilistic Properties" in *Lecture Notes in Computer Science*, Volume 4839/2007 edn, Springer, Berlin, pp. 164-175.

Sankarasubramaniam, Y., Akan, \.B. & Akyildiz, I.F. 2003, "ESRT: event-to-sink reliable transport in wireless sensor networks", *MobiHoc '03: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking \& computing*ACM, New York, NY, USA, pp. 177.

Schurgers, C. & Srivastava, M.B. 2001, "Energy efficient routing in wireless sensor networks", *Military Communications Conference, 2001. MILCOM 2001. Communications for Network-Centric Operations: Creating the Information Force. IEEE*, pp. 357.

Sheng-Lin Wu & Chen, W.-.E. 1996, "The token-bank leaky bucket mechanism for group connections in ATM networks", *Network Protocols, 1996. Proceedings., 1996 International Conference on*, pp. 226.

Shnayder, V., Hempstead, M., Chen, B. & Welsh, M. 2005, 11/10/2005-last update*, Installation Instructions for PowerTOSSIM* [Homepage of Harvard University], [Online]. Available: http://www.eecs.harvard.edu/~shnayder/ptossim/install.html [2010, 06/05/10] .

Shnayder, V., Hempstead, M., Chen, B., Allen, G.W. & Welsh, M. 2004, "Simulating the power consumption of large-scale sensor network applications", *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*ACM, New York, NY, USA, pp. 188.

Slijepcevic, S. & Potkonjak, M. 2001, "Power efficient organization of wireless sensor networks", *Communications, 2001. ICC 2001. IEEE International Conference on*, pp. 472.

Soro, S. & Heinzelman, W.B. 2009, "Cluster head election techniques for coverage preservation in wireless sensor networks", *Ad Hoc Networks,* vol. 7, no. 5, pp. 955-972.

Spadoni, I.M.B., Araujo, R.B. & Marcondes, C. 2009, "Improving QoS in Wireless Sensor Networks through Adaptable Mobile Agents", *INFOCOM Workshops 2009, IEEE*, pp. 1.

Stankovic, J.A. 2006, *Wireless Sensor Networks*, University of Virginia.

Stavrou, E. 2005, 08/06/2005-last update, *Wireless Sensor Networks* [Homepage of Web Hosters], [Online]. Available: http://webhosting.devshed.com/c/a/Web-Hosting-Articles/Wireless-Sensor-Networks-pt-1-Introduction/ [2009, 27/10/2009] .

Striegel, A. & Manimaran, G. 2001, "A scalable approach for DiffServ multicasting", *Communications, 2001. ICC 2001. IEEE International Conference on*, pp. 2327.

Taleghan, M.A., Taherkordi, A., Sharifi, M. & Kim, T. 2007, "A survey of system software for wireless sensor networks", *Proceedings of Future Generation Communication and Networking, Workshop Papers, Vol 2; International Conference on Future Generation Communication Networking*IEEE Comp Soc, , pp. 405.

Thomsen, J. & Husemann, D. 2006, "Evaluating the Use of Motes and TinyOS for a Mobile Sensor Platform", *Parallel and Distributed Computing and Networks: Proceedings of the 24th IASTED International Multi-Conference, .*

TinyOS 2003, *CC1000 Radio Stack Manual*.

Troubleyn, E., De Poorter, E., Moerman, I. & Demeester, P. 2008, "AMoQoSA: Adaptive modular QoS architecture for wireless sensor networks", *2nd International Conference on Sensor Technologies and Applications, SENSORCOMM 2008, August 25, 2008 - August 31*Inst. of Elec. and Elec. Eng. Computer Society, Cap Esterel, France, pp. 172.

Weiser, M. 1993, "Ubiquitous Computing", *Computer,* vol. 26, no. 10, pp. 71-72.

Xia, F. 2008, "QoS challenges and opportunities in wireless sensor/actuator networks", *Sensors,* vol. 8, no. 2, pp. 1099-1110.

Xia, F., Zhao, W., Sun, Y. & Tian, Y. 2007, "Fuzzy logic control based QoS management in wireless sensor/actuator networks", *Sensors,* vol. 7, no. 12, pp. 3179-3191.

Xue, Q. & Ganz, A. 2003, "Ad hoc QoS on-demand routing (AQOR) in mobile ad hoc networks", *Journal of Parallel and Distributed Computing,* vol. 63, no. 2, pp. 154-165.

Yaghmaee, M.H. & Adjeroh, D. 2008, "A Model for Differentiated Service Support in Wireless Multimedia Sensor Networks", *2008 Proceedings of 17th International Conference on Computer Communications and Networks, Vols 1 and 2; IEEE INTERNATIONAL CONFERENCE ON COMPUTER COMMUNICATIONS AND NETWORKS; 17th International Conference on Computer Communications and Networks*IEEE, , pp. 881.

Yick, J., Mukherjee, B. & Ghosal, D. 2008, "Wireless sensor network survey", *Comput.Netw.,* vol. 52, no. 12, pp. 2292-2330.

Younis, M., Akkaya, K., Eltoweissy, M. & Wadaa, A. 2004, "On handling QoS traffic in wireless sensor networks", *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, pp. 10 pp.

Younis, O. & Fahmy, S. 2004, "HEED: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks", *Mobile Computing, IEEE Transactions on,* vol. 3, no. 4, pp. 366-379.

Younis, O. & Fahmy, S. 2005, "An experimental study of routing and data aggregation in sensor networks", *In Proceedings of the IEEE International Workshop on Localized Communication and Topology Protocols for Ad Hoc Networks (IEEE LOCAN*, pp. 50.

Zakaria, A. 2007, *Quality of Service in Wireless Sensor Networks*, Academic survey edn, University of Windsor.

Zhao, Y., Wang, Q., Wang, W., Jiang, D. & Liu, Y. 2009}, "Research on the Priority-based Soft Real-time Task Scheduling in TinyOS", *2009 INTERNATIONAL CONFERENCE ON INFORMATION TECHNOLOGY AND COMPUTER SCIENCE, VOL 1, PROCEEDINGS}*, pp. 562.

Zhou, G., He, T., Krishnamurthy, S. & Stankovic, J.A. 2004, "Impact of radio irregularity on wireless sensor networks", *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*ACM, New York, NY, USA, pp. 125.

Appendix 1 – iHEED About.txt file

Sensor network clustering and data aggregation (iHEED)

Author: Ossama M. Younis (oyounis@cs.purdue.edu).
Affiliation: Purdue University, Department of Computer Science.
First release: January 10, 2005.

The provided code extends TinyOS v.1.1 (http://www.tinyos.net). Below is a description of the modified files and a description of the modifications.

1) tos/interfaces/RouteControl.nc: defines new functions for declaring if the node is a cluster head or is currently in a clustering process.

2) tos/interfaces/EnergyControl.nc: defines functions for for tracking energy consumption of the mote.

3) tos/lib/Route/MultiHop.h: defines CLUSTERING_ON to enable node clustering

4) tos/lib/Route/MultiHopRouter.nc: binds the energy controller and clustering timers to their corresponding components.

5) tos/lib/Route/MultiHopEngineM.nc: energy control and packet capture if the node is a cluster head

6) tos/lib/Route/MultiHopLEPSM.nc: link estimation and parent selection for tree routing. Implements the main clustering process of the HEED algorithm.

7) apps/surge.h: extends the SurgeMsg fields to include the node remaining power, overhead, and the neighboring nodes that are still alive

8) apps/Surge.nc: for energy control and packet interception

9) apps/SurgeM.nc: the main application that uses multi-hop routing and data aggregation. A cluster head aggregates the data it receives before forwarding it on the routing tree. The routing tree is constructed only on the cluster head overlay

Appendix 2 – Output from Building iHEED Application with TOSSIM

```
mkdir -p build/pc
    compiling Surge to a pc binary
ncc -o build/pc/main.exe -g -O0 -I%T/lib/Route -I%T/lib/Queue -I%T/lib/Broadcast -pthread -fnesc-nido-
tosnodes=1000 -fnesc-simulate -Wall -Wshadow -DDEF_TOS_AM_GROUP=0x7d -Wnesc-all -target=pc -
fnesc-cfile=build/pc/app.c -board=micasb -DIDENT_PROGRAM_NAME=\"Surge\" -
DIDENT_USER_ID=\"fraser\" -DIDENT_HOSTNAME=\"fraser-desktop\" -
DIDENT_USER_HASH=0xf3681504L -DIDENT_UNIX_TIME=0x4b757e5bL -
DIDENT_UID_HASH=0xd681ee7fL Surge.nc -lm
In file included from Surge.nc:42:
In component `SurgeM':
SurgeM.nc: In function `SendData':
SurgeM.nc:106: warning: suggest parentheses around assignment used as truth value
In file included from /opt/tinyos-1.x/tos/lib/Route/MultiHopRouter.nc:95,
                 from Surge.nc:43:
In C file:
/opt/tinyos-1.x/tos/lib/Route/MultiHopEngineM.nc: At top level:
/opt/tinyos-1.x/tos/lib/Route/MultiHopEngineM.nc:1: syntax error before `/'
/opt/tinyos-1.x/tos/lib/Route/MultiHopEngineM.nc:1: `$' in identifier
/opt/tinyos-1.x/tos/lib/Route/MultiHopEngineM.nc:1: malformed floating constant
/opt/tinyos-1.x/tos/lib/Route/MultiHopEngineM.nc:1: numeric constant contains digits beyond the radix
/opt/tinyos-1.x/tos/lib/Route/MultiHopEngineM.nc:1: `$' in identifier
In file included from /opt/tinyos-1.x/tos/lib/Route/MultiHopRouter.nc:95,
                 from Surge.nc:43:
In component `MultiHopLEPSM':
/opt/tinyos-1.x/tos/lib/Route/MultiHopLEPSM.nc:53: interface CC1000Control not found
/opt/tinyos-1.x/tos/lib/Route/MultiHopLEPSM.nc: In function `Timer3Task':
/opt/tinyos-1.x/tos/lib/Route/MultiHopLEPSM.nc:695: interface has no command or event named
`SetRFPower'
/opt/tinyos-1.x/tos/lib/Route/MultiHopLEPSM.nc:702: interface has no command or event named
`SetRFPower'
/opt/tinyos-1.x/tos/lib/Route/MultiHopLEPSM.nc: In function `Timer2Task':
/opt/tinyos-1.x/tos/lib/Route/MultiHopLEPSM.nc:750: interface has no command or event named
`SetRFPower'
/opt/tinyos-1.x/tos/lib/Route/MultiHopLEPSM.nc: In function `StdControl.start':
/opt/tinyos-1.x/tos/lib/Route/MultiHopLEPSM.nc:801: interface has no command or event named
`SetRFPower'
```

/opt/tinyos-1.x/tos/lib/Route/MultiHopLEPSM.nc:805: interface has no command or event named `SetRFPower'
/opt/tinyos-1.x/tos/lib/Route/MultiHopLEPSM.nc: In function `EnergyControl.reducePoints':
/opt/tinyos-1.x/tos/lib/Route/MultiHopLEPSM.nc:951: interface has no command or event named `GetRFPower'
/opt/tinyos-1.x/tos/lib/Route/MultiHopLEPSM.nc: In function `EnergyControl.addOverhead':
/opt/tinyos-1.x/tos/lib/Route/MultiHopLEPSM.nc:964: interface has no command or event named `GetRFPower'
/opt/tinyos-1.x/tos/lib/Route/MultiHopLEPSM.nc: In function `SendMsg.sendDone':
/opt/tinyos-1.x/tos/lib/Route/MultiHopLEPSM.nc:1145: interface has no command or event named `GetRFPower'
In file included from Surge.nc:43:
In component `MultiHopRouter':
/opt/tinyos-1.x/tos/lib/Route/MultiHopRouter.nc: At top level:
/opt/tinyos-1.x/tos/lib/Route/MultiHopRouter.nc:103: component CC1000ControlM not found
/opt/tinyos-1.x/tos/lib/Route/MultiHopRouter.nc:136: no match
make: *** [exe0] Error 1

Appendix 3 – eeprom.c versions 1.4

```
1 // $Id: eeprom.c,v 1.4 2005/07/11 05:47:22 jwhui Exp $
    2
    3 /*
        tab:4
    4  * "Copyright (c) 2000-2003 The Regents of the University  of
California.
    5  * All rights reserved.
    6  *
    7  * Permission to use, copy, modify, and distribute this software and
its
    8  * documentation for any purpose, without fee, and without written
agreement is
    9  * hereby granted, provided that the above copyright notice, the
following
   10  * two paragraphs and the author appear in all copies of this software.
   11  *
   12  * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY
PARTY FOR
   13  * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
ARISING OUT
   14  * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE
UNIVERSITY OF
   15  * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
   16  *
   17  * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
   18  * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY
   19  * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED
HEREUNDER IS
   20  * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO
OBLIGATION TO
   21  * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR
MODIFICATIONS."
   22  *
   23  * Copyright (c) 2002-2003 Intel Corporation
   24  * All rights reserved.
   25  *
   26  * This file is distributed under the terms in the attached INTEL-
LICENSE
   27  * file. If you do not find these files, copies can be found by writing
to
   28  * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley,
CA,
   29  * 94704.  Attention:  Intel License Inquiry.
   30  */
   31 /*
   32  *
   33  * Authors:             Philip Levis
   34  * Last updated:11/17/03(Rabin Patra)
```

```
   35  *  - changed fd to local_fd, filename to local_filename
   36  */
   37
   38 /*
   39  *   FILE: eeprom.c
   40  * AUTHOR: Philip Levis <pal@cs.berkeley.edu>
   41  *   DESC: A flat, segmented address space for LOGGER emulation.
   42  */
   43
   44 #include <string.h> // For memcpy(3)
   45 #include <sys/types.h>
   46 #include <sys/stat.h>
   47 #include <fcntl.h>
   48 #include <unistd.h>
   49 #include <sys/mman.h>
   50 #include <errno.h>
   51
   52 static char* local_filename;
   53 static int numMotes = 0;
   54 static int moteSize = 0;
   55 static int initialized = 0;
   56 static int local_fd = -1;
   57
   58 int createEEPROM(char* file, int motes, int eempromBytes) {
   59   int rval;
   60   char val = 0;
   61
   62   uint8_t buf[1024];
   63   uint32_t addr;
   64
   65   memset(buf, 0xff, 1024);
   66
   67   local_filename = file;
   68   numMotes = motes;
   69   moteSize = eempromBytes;
   70
   71   if (initialized) {
   72     dbg(DBG_ERROR, "ERROR: Trying to initialize EEPROM twice.\n");
   73     return -1;
   74   }
   75   local_fd = open(file, O_RDWR | O_CREAT, S_IRWXU | S_IRGRP | S_IROTH);
   76
   77   if (local_fd < 0) {
   78     dbg(DBG_ERROR, "ERROR: Unable to create EEPROM backing store
file.\n");
   79     return -1;
   80   }
   81 /*
   82   rval = (int)lseek(local_fd, (moteSize * numMotes), SEEK_SET);
   83   if (rval < 0) {
   84     dbg(DBG_ERROR, "ERROR: Unable to establish EEPROM of correct
size.\n");
   85   }
   86
   87   rval = write(local_fd, &val, 1);
   88   if (rval < 0) {
```

```
   89     dbg(DBG_ERROR, "ERROR: Unable to establish EEPROM of correct
size.\n");
   90   }
   91 */
   92
   93   rval = (int)lseek(local_fd, (moteSize * numMotes), SEEK_SET);
   94   if (rval < 0) {
   95     dbg(DBG_ERROR, "ERROR: Unable to establish EEPROM of correct
size.\n");
   96   }
   97
   98   rval = read(local_fd, &val, 1);
   99   if (rval < 0) {
  100     dbg(DBG_ERROR, "ERROR: Unable to establish EEPROM of correct
size.\n");
  101   }
  102
  103   if ( !val ) {
  104
  105     rval = (int)lseek(local_fd, 0, SEEK_SET);
  106     if (rval < 0) {
  107       dbg(DBG_ERROR, "ERROR: Unable to establish EEPROM of correct
size.\n");
  108     }
  109
  110     for ( addr = 0; addr <= moteSize * numMotes; addr += 1024 ) {
  111       rval = write(local_fd, buf, 1024);
  112       if (rval < 0) {
  113  dbg(DBG_ERROR, "ERROR: Unable to establish EEPROM of correct
size.\n");
  114       }
  115     }
  116   }
  117
  118   initialized = 1;
  119
  120   return local_fd;
  121 }
  122
  123 int anonymousEEPROM(int fnumMotes, int eepromSize) {
  124   int filedes;
  125   filedes = createEEPROM("/tmp/anonymous", fnumMotes, eepromSize);
  126   if (filedes >= 0) {
  127     unlink("/tmp/anonymous");
  128     return 0;
  129   }
  130   else {
  131     dbg(DBG_ERROR, "ERROR: Unable to create anonymous EEPROM
region.\n");
  132     return -1;
  133   }
  134 }
  135
  136 int namedEEPROM(char* name, int fnumMotes, int eepromSize) {
  137   int filedes = createEEPROM(name, fnumMotes, eepromSize);
  138   if (filedes >= 0) {
  139     return 0;
```

```
  140   }
  141   else {
  142     dbg(DBG_ERROR, "ERROR: Unable to create named EEPROM region:
%s.\n", name);
  143     return -1;
  144   }
  145 }
  146
  147 int readEEPROM(char* buffer, int mote, int offset, int length) {
  148   // Sanity check arguments; don't want to corrupt data.
  149   if (mote > numMotes || mote < 0) {
  150     dbg(DBG_ERROR, "ERROR: Tried to read EEPROM of mote %i when it was
initialized for only %i motes.\n", mote, numMotes);
  151     return -1;
  152   }
  153   else if ((offset + length) > moteSize) {
  154     dbg(DBG_ERROR, "ERROR: Tried to read EEPROM address 0x%x of mote
when its max EEPROM address is 0x%x.\n", (offset + length), moteSize);
  155     return -1;
  156   }
  157   else if (length < 0 || offset < 0) {
  158     dbg(DBG_ERROR, "ERROR: Both length and offset for EEPROM reads must
be > 0.\n");
  159     return -1;
  160   }
  161   else {
  162     int rval;
  163     int startOffset = mote * moteSize;
  164     int seekedOffset = startOffset + offset;
  165     rval = lseek(local_fd, seekedOffset, SEEK_SET);
  166     if (rval < 0) {
  167       dbg(DBG_ERROR, "ERROR: Seek in EEPROM for read failed.\n");
  168     }
  169     rval = read(local_fd, buffer, length);
  170     if (rval <= 0) {
  171       dbg(DBG_ERROR, "ERROR: Read for %i from EEPROM failed: %s.\n",
length, strerror(errno));
  172     }
  173     return 0;
  174   }
  175 }
  176
  177 int writeEEPROM(char* buffer, int mote, int offset, int length) {
  178   // Sanity check arguments; don't want to corrupt data.
  179   if (mote > numMotes || mote < 0) {
  180     dbg(DBG_ERROR, "ERROR: Tried to write EEPROM of mote %i when it was
initialized for only %i motes.\n", mote, numMotes);
  181     return -1;
  182   }
  183   else if ((offset + length) > moteSize) {
  184     dbg(DBG_ERROR, "ERROR: Tried to write EEPROM address 0x%x of mote
when its max EEPROM address is 0x%x.\n", (offset + length), moteSize);
  185     return -1;
  186   }
  187   else if (length < 0 || offset < 0) {
  188     dbg(DBG_ERROR, "ERROR: Both length and offset for EEPROM write must
be > 0.\n");
```

```
  189     return -1;
  190   }
  191   else {
  192     int rval;
  193     int startOffset = mote * moteSize;
  194     int seekedOffset = startOffset + offset;
  195     rval = lseek(local_fd, seekedOffset, SEEK_SET);
  196     if (rval < 0) {
  197       dbg(DBG_ERROR, "ERROR: Seek in EEPROM for write failed: %s.\n",
strerror(errno));
  198     }
  199     rval = write(local_fd, buffer, length);
  200     if (rval <= 0) {
  201       dbg(DBG_ERROR, "ERROR: Write to EEPROM failed: %s.\n",
strerror(errno));
  202     }
  203     return 0;
  204   }
  205 }
  206
  207 int syncEEPROM() {
  208   return fsync(local_fd);
  209 }
```

Appendix 4 – eeprom.c version 1.3

```
1 // $Id: eeprom.c,v 1.3 2004/04/27 23:24:09 scipio Exp $
    2
    3 /*
        tab:4
    4  * "Copyright (c) 2000-2003 The Regents of the University  of
California.
    5  * All rights reserved.
    6  *
    7  * Permission to use, copy, modify, and distribute this software and
its
    8  * documentation for any purpose, without fee, and without written
agreement is
    9  * hereby granted, provided that the above copyright notice, the
following
   10  * two paragraphs and the author appear in all copies of this software.
   11  *
   12  * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY
PARTY FOR
   13  * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
ARISING OUT
   14  * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE
UNIVERSITY OF
   15  * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
   16  *
   17  * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
   18  * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY
   19  * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED
HEREUNDER IS
   20  * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO
OBLIGATION TO
   21  * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR
MODIFICATIONS."
   22  *
   23  * Copyright (c) 2002-2003 Intel Corporation
   24  * All rights reserved.
   25  *
   26  * This file is distributed under the terms in the attached INTEL-
LICENSE
   27  * file. If you do not find these files, copies can be found by writing
to
   28  * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley,
CA,
   29  * 94704.  Attention:  Intel License Inquiry.
   30  */
   31 /*
   32  *
   33  * Authors:             Philip Levis
   34  * Last updated:11/17/03(Rabin Patra)
```

```
 *  - changed fd to local_fd, filename to local_filename
 */

/*
 *   FILE: eeprom.c
 * AUTHOR: Philip Levis <pal@cs.berkeley.edu>
 *   DESC: A flat, segmented address space for LOGGER emulation.
 */

#include <string.h> // For memcpy(3)
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <errno.h>

static char* local_filename;
static int numMotes = 0;
static int moteSize = 0;
static int initialized = 0;
static int local_fd = -1;

int createEEPROM(char* file, int motes, int eempromBytes) {
  int rval;
  char val = 0;

  local_filename = file;
  numMotes = motes;
  moteSize = eempromBytes;

  if (initialized) {
    dbg(DBG_ERROR, "ERROR: Trying to initialize EEPROM twice.\n");
    return -1;
  }
  local_fd = open(file, O_RDWR | O_CREAT, S_IRWXU | S_IRGRP | S_IROTH);

  if (local_fd < 0) {
    dbg(DBG_ERROR, "ERROR: Unable to create EEPROM backing store
file.\n");
    return -1;
  }

  rval = (int)lseek(local_fd, (moteSize * numMotes), SEEK_SET);
  if (rval < 0) {
    dbg(DBG_ERROR, "ERROR: Unable to establish EEPROM of correct
size.\n");
  }

  rval = write(local_fd, &val, 1);
  if (rval < 0) {
    dbg(DBG_ERROR, "ERROR: Unable to establish EEPROM of correct
size.\n");
  }
  initialized = 1;

  return local_fd;
```

```
   89 }
   90
   91 int anonymousEEPROM(int fnumMotes, int eepromSize) {
   92   int filedes;
   93   filedes = createEEPROM("/tmp/anonymous", fnumMotes, eepromSize);
   94   if (filedes >= 0) {
   95     unlink("/tmp/anonymous");
   96     return 0;
   97   }
   98   else {
   99     dbg(DBG_ERROR, "ERROR: Unable to create anonymous EEPROM
region.\n");
  100     return -1;
  101   }
  102 }
  103
  104 int namedEEPROM(char* name, int fnumMotes, int eepromSize) {
  105   int filedes = createEEPROM(name, fnumMotes, eepromSize);
  106   if (filedes >= 0) {
  107     return 0;
  108   }
  109   else {
  110     dbg(DBG_ERROR, "ERROR: Unable to create named EEPROM region:
%s.\n", name);
  111     return -1;
  112   }
  113 }
  114
  115 int readEEPROM(char* buffer, int mote, int offset, int length) {
  116   // Sanity check arguments; don't want to corrupt data.
  117   if (mote > numMotes || mote < 0) {
  118     dbg(DBG_ERROR, "ERROR: Tried to read EEPROM of mote %i when it was
initialized for only %i motes.\n", mote, numMotes);
  119     return -1;
  120   }
  121   else if ((offset + length) > moteSize) {
  122     dbg(DBG_ERROR, "ERROR: Tried to read EEPROM address 0x%x of mote
when its max EEPROM address is 0x%x.\n", (offset + length), moteSize);
  123     return -1;
  124   }
  125   else if (length < 0 || offset < 0) {
  126     dbg(DBG_ERROR, "ERROR: Both length and offset for EEPROM reads must
be > 0.\n");
  127     return -1;
  128   }
  129   else {
  130     int rval;
  131     int startOffset = mote * moteSize;
  132     int seekedOffset = startOffset + offset;
  133     rval = lseek(local_fd, seekedOffset, SEEK_SET);
  134     if (rval < 0) {
  135       dbg(DBG_ERROR, "ERROR: Seek in EEPROM for read failed.\n");
  136     }
  137     rval = read(local_fd, buffer, length);
  138     if (rval <= 0) {
  139       dbg(DBG_ERROR, "ERROR: Read for %i from EEPROM failed: %s.\n",
length, strerror(errno));
```

```
  140     }
  141     return 0;
  142   }
  143 }
  144
  145 int writeEEPROM(char* buffer, int mote, int offset, int length) {
  146   // Sanity check arguments; don't want to corrupt data.
  147   if (mote > numMotes || mote < 0) {
  148     dbg(DBG_ERROR, "ERROR: Tried to write EEPROM of mote %i when it was
initialized for only %i motes.\n", mote, numMotes);
  149     return -1;
  150   }
  151   else if ((offset + length) > moteSize) {
  152     dbg(DBG_ERROR, "ERROR: Tried to write EEPROM address 0x%x of mote
when its max EEPROM address is 0x%x.\n", (offset + length), moteSize);
  153     return -1;
  154   }
  155   else if (length < 0 || offset < 0) {
  156     dbg(DBG_ERROR, "ERROR: Both length and offset for EEPROM write must
be > 0.\n");
  157     return -1;
  158   }
  159   else {
  160     int rval;
  161     int startOffset = mote * moteSize;
  162     int seekedOffset = startOffset + offset;
  163     rval = lseek(local_fd, seekedOffset, SEEK_SET);
  164     if (rval < 0) {
  165       dbg(DBG_ERROR, "ERROR: Seek in EEPROM for write failed: %s.\n",
strerror(errno));
  166     }
  167     rval = write(local_fd, buffer, length);
  168     if (rval <= 0) {
  169       dbg(DBG_ERROR, "ERROR: Write to EEPROM failed: %s.\n",
strerror(errno));
  170     }
  171     return 0;
  172   }
  173 }
  174
  175 int syncEEPROM() {
  176   return fsync(local_fd);
  177 }
```

Appendix 5 – original SurgeM.nc file

```
1 // $Id: SurgeM.nc,v 1.9 2005/02/28 22:23:34 philipb Exp $
    2
    3 /*
        tab:4
    4  * "Copyright (c) 2000-2003 The Regents of the University  of
California.
    5  * All rights reserved.
    6  *
    7  * Permission to use, copy, modify, and distribute this software and
its
    8  * documentation for any purpose, without fee, and without written
agreement is
    9  * hereby granted, provided that the above copyright notice, the
following
   10  * two paragraphs and the author appear in all copies of this software.
   11  *
   12  * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY
PARTY FOR
   13  * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
ARISING OUT
   14  * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE
UNIVERSITY OF
   15  * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
   16  *
   17  * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
   18  * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY
   19  * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED
HEREUNDER IS
   20  * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO
OBLIGATION TO
   21  * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR
MODIFICATIONS."
   22  *
   23  * Copyright (c) 2002-2003 Intel Corporation
   24  * All rights reserved.
   25  *
   26  * This file is distributed under the terms in the attached INTEL-
LICENSE
   27  * file. If you do not find these files, copies can be found by writing
to
   28  * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley,
CA,
   29  * 94704.  Attention:  Intel License Inquiry.
   30  */
   31
   32 includes Surge;
   33 includes SurgeCmd;
```

```
34
35 /*
36  *  Data gather application
37  */
38
39 module SurgeM {
40   provides {
41     interface StdControl;
42   }
43   uses {
44     interface ADC;
45     interface Timer;
46     interface Leds;
47     interface StdControl as Sounder;
48     interface Send;
49     interface Receive as Bcast;
50     interface RouteControl;
51   }
52 }
53
54 implementation {
55
56   enum {
57     TIMER_GETADC_COUNT = 1,            // Timer ticks for ADC
58     TIMER_CHIRP_COUNT = 10,            // Timer on/off chirp count
59   };
60
61   bool sleeping;                // application command state
62   bool focused;
63   bool rebroadcast_adc_packet;
64
65   TOS_Msg gMsgBuffer;
66   norace uint16_t gSensorData;          // protected by gfSendBusy flag
67   bool gfSendBusy;
68
69
70   int timer_rate;
71   int timer_ticks;
72
/*****************************************************************
73    * Initialization
74
*****************************************************************/
75
76   static void initialize() {
77     timer_rate = INITIAL_TIMER_RATE;
78     atomic gfSendBusy = FALSE;
79     sleeping = FALSE;
80     rebroadcast_adc_packet = FALSE;
81     focused = FALSE;
82   }
83
84   task void SendData() {
85     SurgeMsg *pReading;
86     uint16_t Len;
87     dbg(DBG_USR1, "SurgeM: Sending sensor reading\n");
88
```

```
 89     if (pReading = (SurgeMsg *)call Send.getBuffer(&gMsgBuffer,&Len)) {
 90       pReading->type = SURGE_TYPE_SENSORREADING;
 91       pReading->parentaddr = call RouteControl.getParent();
 92       pReading->reading = gSensorData;
 93
 94       if ((call Send.send(&gMsgBuffer,sizeof(SurgeMsg))) != SUCCESS)
 95   atomic gfSendBusy = FALSE;
 96     }
 97
 98   }
 99
100   command result_t StdControl.init() {
101     call Leds.init();
102     initialize();
103     return SUCCESS;
104   }
105
106   command result_t StdControl.start() {
107     return call Timer.start(TIMER_REPEAT, timer_rate);
108     return SUCCESS;
109   }
110
111   command result_t StdControl.stop() {
112     return call Timer.stop();
113   }
114
115
/*****************************************************************
116    * Commands and events
117
*****************************************************************/
118
119   event result_t Timer.fired() {
120     dbg(DBG_USR1, "SurgeM: Timer fired\n");
121     timer_ticks++;
122     if (timer_ticks % TIMER_GETADC_COUNT == 0) {
123       call ADC.getData();
124     }
125     // If we're the focused node, chirp
126     if (focused && timer_ticks % TIMER_CHIRP_COUNT == 0) {
127       call Sounder.start();
128     }
129     // If we're the focused node, chirp
130     if (focused && timer_ticks % TIMER_CHIRP_COUNT == 1) {
131       call Sounder.stop();
132     }
133     return SUCCESS;
134   }
135
136   async event result_t ADC.dataReady(uint16_t data) {
137     //SurgeMsg *pReading;
138     //uint16_t Len;
139     dbg(DBG_USR1, "SurgeM: Got ADC reading: 0x%x\n", data);
140     atomic {
141       if (!gfSendBusy) {
142   gfSendBusy = TRUE;
143   gSensorData = data;
```

```
  144   post SendData();
  145       }
  146     }
  147     return SUCCESS;
  148   }
  149
  150   event result_t Send.sendDone(TOS_MsgPtr pMsg, result_t success) {
  151     dbg(DBG_USR2, "SurgeM: output complete 0x%x\n", success);
  152     //call Leds.greenToggle();
  153     atomic gfSendBusy = FALSE;
  154     return SUCCESS;
  155   }
  156
  157
  158   /* Command interpreter for broadcasts
  159    *
  160    */
  161
  162   event TOS_MsgPtr Bcast.receive(TOS_MsgPtr pMsg, void* payload,
uint16_t payloadLen) {
  163     SurgeCmdMsg *pCmdMsg = (SurgeCmdMsg *)payload;
  164
  165     dbg(DBG_USR2, "SurgeM: Bcast  type 0x%02x\n", pCmdMsg->type);
  166
  167     if (pCmdMsg->type == SURGE_TYPE_SETRATE) {       // Set timer rate
  168       timer_rate = pCmdMsg->args.newrate;
  169       dbg(DBG_USR2, "SurgeM: set rate %d\n", timer_rate);
  170       call Timer.stop();
  171       call Timer.start(TIMER_REPEAT, timer_rate);
  172
  173     } else if (pCmdMsg->type == SURGE_TYPE_SLEEP) {
  174       // Go to sleep - ignore everything until a SURGE_TYPE_WAKEUP
  175       dbg(DBG_USR2, "SurgeM: sleep\n");
  176       sleeping = TRUE;
  177       call Timer.stop();
  178       call Leds.greenOff();
  179       call Leds.yellowOff();
  180
  181     } else if (pCmdMsg->type == SURGE_TYPE_WAKEUP) {
  182       dbg(DBG_USR2, "SurgeM: wakeup\n");
  183
  184       // Wake up from sleep state
  185       if (sleeping) {
  186   initialize();
  187         call Timer.start(TIMER_REPEAT, timer_rate);
  188   sleeping = FALSE;
  189       }
  190
  191     } else if (pCmdMsg->type == SURGE_TYPE_FOCUS) {
  192       dbg(DBG_USR2, "SurgeM: focus %d\n", pCmdMsg->args.focusaddr);
  193       // Cause just one node to chirp and increase its sample rate;
  194       // all other nodes stop sending samples (for demo)
  195       if (pCmdMsg->args.focusaddr == TOS_LOCAL_ADDRESS) {
  196   // OK, we're focusing on me
  197   focused = TRUE;
  198   call Sounder.init();
  199   call Timer.stop();
```

```
200   call Timer.start(TIMER_REPEAT, FOCUS_TIMER_RATE);
201       } else {
202   // Focusing on someone else
203   call Timer.stop();
204   call Timer.start(TIMER_REPEAT, FOCUS_NOTME_TIMER_RATE);
205       }
206
207     } else if (pCmdMsg->type == SURGE_TYPE_UNFOCUS) {
208       // Return to normal after focus command
209       dbg(DBG_USR2, "SurgeM: unfocus\n");
210       focused = FALSE;
211       call Sounder.stop();
212       call Timer.stop();
213       call Timer.start(TIMER_REPEAT, timer_rate);
214     }
215     return pMsg;
216   }
217
218 }
219
220
```

Appendix 6 – iHEED version of SurgeM.nc

```
// $Id: SurgeM.nc,v 1.7.2.3 2003/08/18 22:09:37 cssharp Exp $


/*                                                                      tab:4
 * "Copyright (c) 2000-2003 The Regents of the University  of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
```

```
includes Surge;
includes SurgeCmd;

/*
 *  Data gather application
 */

module SurgeM {
 provides {
   interface StdControl;
 }
 uses {
   interface ADC;
   interface Timer;
   interface Leds;
```

```
    interface StdControl as Sounder;
    interface Send;
    interface Receive as Bcast;
    interface RouteControl;
        /*** HEED: clustering and control ******/
        interface EnergyControl;
        interface Intercept as InterceptSurgeMsg;
  }
}

implementation {

  enum {
    TIMER_GETADC_COUNT = 1,          // Timer ticks for ADC
    TIMER_CHIRP_COUNT = 10,          // Timer on/off chirp count
  };

  bool sleeping;                     // application command state
  bool focused;
  bool rebroadcast_adc_packet;

  TOS_Msg gMsgBuffer;
  norace uint16_t gSensorData;             // protected by gfSendBusy flag
  bool gfSendBusy;
```

```
int timer_rate;
int timer_ticks;

/*** HEED: clustering and control ******/
#define COLLECT_COUNT 1
uint16_t numCollectedPoints, numSentPoints, motesAlive, sendMotesAlive;
int CH_timer_ticks, reportNumber;
/**************************************/

/*****************************************************************************
 * Initialization
 *****************************************************************************/

static void initialize() {
  timer_rate = INITIAL_TIMER_RATE;
  atomic gfSendBusy = FALSE;
  sleeping = FALSE;
  rebroadcast_adc_packet = FALSE;
  focused = FALSE;

        /*** HEED: clustering and control ******/
        numCollectedPoints = numSentPoints = 0;
        motesAlive = sendMotesAlive = 0;
        CH_timer_ticks = 0;
        reportNumber = 0;
```

```
        /****************************************/
 }

 task void SendData() {
  SurgeMsg *pReading;
  uint16_t Len;

  dbg(DBG_USR1, "SurgeM: Sending sensor reading\n");

  if (pReading = (SurgeMsg *)call Send.getBuffer(&gMsgBuffer,&Len)) {
   pReading->type = SURGE_TYPE_SENSORREADING;
   pReading->parentaddr = call RouteControl.getParent();
        //pReading->reading = gSensorData;
        pReading->reading = reportNumber;          // use gSensorData for real data collection

        /***** HEED code **********/
        pReading->remPower = call EnergyControl.getRemainingPoints();          // remaining
power
        pReading->overhead = call EnergyControl.getOverhead();          // overhead so
far
        pReading->nPoints = numSentPoints;
       // # collected readings
        pReading->motesAlive = sendMotesAlive;
       // which motes are alive
        /*************************/

   if ((call Send.send(&gMsgBuffer,sizeof(SurgeMsg))) != SUCCESS)
```

```
                    atomic gfSendBusy = FALSE;
  }
}

command result_t StdControl.init() {
  initialize();
  return SUCCESS;
}

command result_t StdControl.start() {
  return call Timer.start(TIMER_REPEAT, timer_rate);
  return SUCCESS;
}

command result_t StdControl.stop() {
  return call Timer.stop();
}

/****************************************************************************
 * Commands and events
 ****************************************************************************/

event result_t Timer.fired() {

        // make sure there is still some power left
```

```
        if (!(call EnergyControl.isAlive())) {
                call StdControl.stop();
                return SUCCESS;
        }

#ifdef CLUSTERING_ON
        // Case 1. Clustering is being applied ...
        // generate your locally sensed data
        if (TOS_LOCAL_ADDRESS != 0) {
                numCollectedPoints++;
                motesAlive |= (1 << (TOS_LOCAL_ADDRESS-1));
        }
        CH_timer_ticks++;
        // If no parent is available, continue holding the data.
        if (call RouteControl.getParent() == 0xFFFF) {
                return SUCCESS;
        }
        // If cluster head, send aggregated data
        if (call RouteControl.isClusterHead()) {
                if (CH_timer_ticks % COLLECT_COUNT == 0) {
                        atomic {
                                numSentPoints = numCollectedPoints;
                                sendMotesAlive = motesAlive;
                                motesAlive = 0;
                                if (TOS_LOCAL_ADDRESS != 0)
```

```
                                        numCollectedPoints = 0;
                                reportNumber++;
                                if (!gfSendBusy) {
                                        gfSendBusy = TRUE;
                                        post SendData();
                                }
                        }
                }
        }
        // If not cluster head, send your reading
        else {
                atomic {
                        numSentPoints = numCollectedPoints;
                        sendMotesAlive = motesAlive;
                        motesAlive = 0;
                        if (TOS_LOCAL_ADDRESS != 0)
                                numCollectedPoints = 0;
                        reportNumber++;
                        if (!gfSendBusy) {
                                gfSendBusy = TRUE;
                                post SendData();
                        }
                }
        }
#else
```

```
		// Case 2. Clustering is not applied
		// Base station send aggregated data
		// Regular motes simply forward their readings.
		atomic {
			if (TOS_LOCAL_ADDRESS == 0)  {
				//numSentPoints++;
				atomic {
					numSentPoints += numCollectedPoints;
					numCollectedPoints = 0;
				}
			}
			else
				numSentPoints = 1;
			reportNumber++;
			if (!gfSendBusy) {
				gfSendBusy = TRUE;
				post SendData();
			}
		}
		call Leds.redToggle();
#endif
/*    dbg(DBG_USR1, "SurgeM: Timer fired\n");
   timer_ticks++;
   if (timer_ticks % TIMER_GETADC_COUNT == 0) {
     call ADC.getData();
```

```
  }
  // If we're the focused node, chirp
  if (focused && timer_ticks % TIMER_CHIRP_COUNT == 0) {
    call Sounder.start();
  }
  // If we're the focused node, chirp
  if (focused && timer_ticks % TIMER_CHIRP_COUNT == 1) {
    call Sounder.stop();
  }*/
  return SUCCESS;
}

async event result_t ADC.dataReady(uint16_t data) {
  //SurgeMsg *pReading;
  //uint16_t Len;
  dbg(DBG_USR1, "SurgeM: Got ADC reading: 0x%x\n", data);
  atomic {
    if (!gfSendBusy) {
          gfSendBusy = TRUE;
          gSensorData = data;
          post SendData();
    }
  }
  return SUCCESS;
}
```

```
event result_t Send.sendDone(TOS_MsgPtr pMsg, result_t success) {
  dbg(DBG_USR2, "SurgeM: output complete 0x%x\n", success);
  //call Leds.greenToggle();
  atomic gfSendBusy = FALSE;
  return SUCCESS;
}


/* Command interpreter for broadcasts
 *
 */

event TOS_MsgPtr Bcast.receive(TOS_MsgPtr pMsg, void* payload, uint16_t payloadLen) {
  SurgeCmdMsg *pCmdMsg = (SurgeCmdMsg *)payload;

  dbg(DBG_USR2, "SurgeM: Bcast  type 0x%02x\n", pCmdMsg->type);

  if (pCmdMsg->type == SURGE_TYPE_SETRATE) {       // Set timer rate
    timer_rate = pCmdMsg->args.newrate;
    dbg(DBG_USR2, "SurgeM: set rate %d\n", timer_rate);
    call Timer.stop();
    call Timer.start(TIMER_REPEAT, timer_rate);

  } else if (pCmdMsg->type == SURGE_TYPE_SLEEP) {
```

```
  // Go to sleep - ignore everything until a SURGE_TYPE_WAKEUP
  dbg(DBG_USR2, "SurgeM: sleep\n");
  sleeping = TRUE;
  call Timer.stop();
  call Leds.greenOff();
  call Leds.yellowOff();

} else if (pCmdMsg->type == SURGE_TYPE_WAKEUP) {
  dbg(DBG_USR2, "SurgeM: wakeup\n");

  // Wake up from sleep state
  if (sleeping) {
        initialize();
    call Timer.start(TIMER_REPEAT, timer_rate);
        sleeping = FALSE;
  }

} else if (pCmdMsg->type == SURGE_TYPE_FOCUS) {
  dbg(DBG_USR2, "SurgeM: focus %d\n", pCmdMsg->args.focusaddr);
  // Cause just one node to chirp and increase its sample rate;
  // all other nodes stop sending samples (for demo)
  if (pCmdMsg->args.focusaddr == TOS_LOCAL_ADDRESS) {
        // OK, we're focusing on me
        focused = TRUE;
        call Sounder.init();
```

```
            call Timer.stop();
            call Timer.start(TIMER_REPEAT, FOCUS_TIMER_RATE);
        } else {
            // Focusing on someone else
            call Timer.stop();
            call Timer.start(TIMER_REPEAT, FOCUS_NOTME_TIMER_RATE);
        }

    } else if (pCmdMsg->type == SURGE_TYPE_UNFOCUS) {
        // Return to normal after focus command
        dbg(DBG_USR2, "SurgeM: unfocus\n");
        focused = FALSE;
        call Sounder.stop();
        call Timer.stop();
        call Timer.start(TIMER_REPEAT, timer_rate);
    }
    return pMsg;
  }

/*****************************************
 *                  Intercept a Surge Message
 *                          Collect and aggregate the data
 *                          if you are a cluster head
 *****************************************/
```

```
 event result_t InterceptSurgeMsg.intercept(TOS_MsgPtr msg, void* payload, uint16_t payloadLen) {
   SurgeMsg *sMsg = (SurgeMsg *)payload;

#ifdef CLUSTERING_ON
        if (call RouteControl.isClusterHead()) {
                if (sMsg->type == SURGE_TYPE_SENSORREADING) {
                        numCollectedPoints += sMsg->nPoints;
                        motesAlive |= sMsg->motesAlive;
                }
        }
#else
        if (sMsg->type == SURGE_TYPE_SENSORREADING && TOS_LOCAL_ADDRESS == 0) {
                numCollectedPoints++;
        }
#endif
        call Leds.yellowToggle();
        return SUCCESS;
 }

}
```

Appendix 8 – sample of reliability measurements

Surge

222: Sending Sensed Data

178: Sending Sensed Data

123: Sending Sensed Data

224: Sending Sensed Data

226: Sending Sensed Data

285: Sending Sensed Data

355: Sending Sensed Data

368: Sending Sensed Data

20: Sending Sensed Data

0: BaseStation: Received Readings

iHEED

206: CLUSTERMEMBER: sending senseddata to cluster-head

54: CLUSTERHEAD: sending readings

206: CLUSTERMEMBER: sending senseddata to cluster-head

54: CLUSTERHEAD: sending readings

206: CLUSTERMEMBER: sending senseddata to cluster-head

54: CLUSTERHEAD: sending readings

206: CLUSTERMEMBER: sending senseddata to cluster-head

54: CLUSTERHEAD: sending readings

206: CLUSTERMEMBER: sending senseddata to cluster-head

Appendix 9 – Sample of Power Analysis

Surge

Mote 98, cpu total: 7374.463931

Mote 98, radio total: 12729.645888

Mote 98, adc total: 0.000000

Mote 98, leds total: 0.000000

Mote 98, sensor total: 1232.053296

Mote 98, eeprom total: 0.000000

Mote 98, cpu_cycle total: 0.000000

Mote 98, Total energy: 21336.163115

iHEED

Mote 86, cpu total: 7427.074340

Mote 86, radio total: 12584.095855

Mote 86, adc total: 0.000000

Mote 86, leds total: 5042.860553

Mote 86, sensor total: 1240.842928

Mote 86, eeprom total: 0.000000

Mote 86, cpu_cycle total: 0.000000

Mote 86, Total energy: 26294.873677