Imperial College, London
Department of Computing

Bio-Inspired Tools for a Distributed Wireless Sensor Network Operating System.

Michael Breza

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of Imperial College, London April 26th, 2013

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given in the bibliography.

Abstract

The problem which I address in this thesis is to find a way to organise and manage a network of wireless sensor nodes using a minimal amount of communication. To find a solution I explore the use of Bio-inspired protocols to enable WSN management while maintaining a low communication overhead. Wireless Sensor Networks (WSNs) are loosely coupled distributed systems comprised of low-resource, battery powered sensor nodes. The largest problem with WSN management is that communication is the largest consumer of a sensor node's energy. WSN management systems need to use as little communication as possible to prolong their operational lifetimes. This is the Wireless Sensor Network Management Problem. This problem is compounded because current WSN management systems glue together unrelated protocols to provide system services causing inter-protocol interference. Bio-inspired protocols provide a good solution because they enable the nodes to self-organise, use local area communication, and can combine their communication in an intelligent way with minimal increase in communication. I present a combined protocol and MAC scheduler to enable multiple service protocols to function in a WSN at the same time without causing inter-protocol interference. The scheduler is throughput optimal as long as the communication requirements of all of the protocols remain within the communication capacity of the network. I show that the scheduler improves a dissemination protocol's performance by 35%. A bio-inspired synchronisation service is presented which enables wireless sensor nodes to self organise and provide a time service. Evaluation of the protocol shows an 80% saving in communication over similar bio-inspired synchronisation approaches. I then add an information dissemination protocol, without significantly increasing communication. This is achieved through the ability of our bio-inspired algorithms to combine their communication in an intelligent way so that they are able to offer multiple services without requiring a great deal of inter-node communication.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Wireless Sensor Networks

Science has long been synonymous with exploration and discovery. Key to scientific exploration has been the tools developed to make new observations and gather new information about the world around us. The development of the lens led to the invention of the telescope, which allowed observation and discovery about planets and stars far from our own and only previously observed as specs of light in the sky. From the same lens technology came the microscope, which enabled the study of our immediate world at a scale not perceivable with the naked eye. Today a new tool is being developed which will allow us to observe phenomenon that occur all around us, but are impossible to observe or measure because they take place at many different points or locations at the same time. This is a spatio-temporal space, and is within the realm of distributed phenomenon such as the flocking of birds, the spreading of rumours, or the spread of pollutants like smoke around a room.

Throughout this thesis I use a simple example to illustrate our ideas. Imagine being in a smoke filled bar one evening (back when it was legal to smoke in bars). If you are a non-smoker, chances are that you do not particularly like this smoke, it may make your clothes smell, or your eyes water. The question becomes, where can I sit with my non-smoking friends to give ourselves as little exposure to the smoke as possible? If you had a machine to measure the level

of pollutants in the air, you could wander around the bar taking readings at different locations in the bar, and find out where the cleanest air is. The problem is that the time at which you take each measurement will be different for each location. As smoke floats on the air, and is in constant motion as people move around, a location with clean air at the moment you take a measurement, may not have clean air for very long. A better way would be to have many different pollution detectors, and measure various locations around the bar at the same time. If the pollution detectors are hand-held, you would need a lot of helpers, and each person would have an affect on the motion of the air being measured. A mathematical model describing this sort of phenomenon is very complex. The model would need to take account of the motion of each smoke particle in the room. This sort of analysis would require vast computing resources. The answer, of which part of the bar has the cleanest air, is very difficult to get.

There is a currently evolving solution to this problem, a way to explore these sorts of distributed spatio-temporal phenomenon. Very small battery powered computers equipped with very small sensors can be distributed in a given environment to take synchronised measurements, and communicate this data back to a central point using wireless radios. The data can then be used to understand how a complex distributed phenomenon like smoke moving about a bar, actually happens. The possibilities for this technology are exciting. Like the telescope and the microscope before, it allows us to observe phenomenon which happen right in front of us, but are difficult, if not impossible to measure and therefore understand.

This new technology is referred to as a Wireless Sensor Network (WSN from here, WSNs in the plural form). WSNs have the ability to expose distributed phenomenon occurring around us, help us to understand them, and possibly even learn to emulate and control them. They allow us to observe and collect data from multiple concurrent events through their ability to measure several locations at the same time. WSNs provide for us a platform to experiment with distributed phenomenon.

The problem that is focused on in this work is how to enable small, resource-constrained sensor nodes to organise themselves into a coherent system and provide services to an application. The challenge posed by this problem is to find a way to enable self-organisation whilst not unduly

taxing the limited energy resources of the sensor nodes. This is the WSN management problem. Our solution is to use bio-inspired protocols to make the sensor nodes organise themselves. Bio-inspired protocols are chosen because they can be used to control distributed systems of loosely coupled, low capacity devices and get good performance within the constraints of a WSNs limited resources.

1.2 The Problems with Wireless Sensor Network Management.

One of the outstanding problems preventing the wide-scale adoption of WSN is the lack of a management solution with which to make coherent WSN systems [TGC06] [1]. I define a system as a collection of WSN nodes which work together to provide services which no individual sensor node could provide on its own. What is needed is a distributed operating system that provides the organisation and management functions that current operating systems provide on mobile phones and desktop computers. It is this operating system type of resource management that I refer to when we use the term wireless sensor network management (WSN management).

I also take the view in this thesis that the most desirable WSN system consists of just the sensor nodes, and does not rely on any larger, more resource rich nodes such as data collecting base-stations for WSN system organisation. Base-stations can not be depended upon for a variety of reasons: they can be mobile units which visit the WSN infrequently, they can fail due to unforeseen circumstances like vandalism or curious animals, or can cease to function properly from situations such as the scale of the network growing too large. Therefore, it is unwise to have any single points of failure that the rest of the network relies upon.

WSNs are made out of a potentially large number of radio connected computer devices with finite energy, low capability processors, and a small amount of memory when compared to desktop

[1] The continued lack of WSN management solutions is a point of which I had two WSN luminaries agree with at two separate events. David Culler agreed that there was a lack of work done in the area of WSN management at ISPN 2011, and John Stankovic agreed to the same problem at SensorNets 2013. Hearsay does not make the best evidence, but the opinions of some (such as the gentlemen mentioned above) are certainly worthy of attention.

computers or mobile smart-phones. The fact that they are distributed means that communication is the most important element in system creation and maintenance. Communication is limited in WSN, both in terms of the energy costs for the nodes, and the communication capacity of the radio links used to form the network. The central problem of this thesis is this requirement for communication that is needed by WSN management, and the inability of battery powered sensor nodes to communicate a great deal. We refer to this dilemma as the WSN management problem, and we make it a central consideration in our work.

It is my view that the reason that WSN system development has been slow is that WSNs represent a new class of distributed computing systems. On its own, one sensor node cannot give much information, or do very much. When grouped with a large number of other sensor nodes, a WSN can perform distributed computation such as data aggregation, form routes for data forwarding, and solve other interesting distributed problems.

WSN systems have some unique aspects that give them a set of challenges that are distinct from other distributed computer systems. WSNs, therefore, need a new approach to system design. Their diminutive size and cost allows WSNs to be deployed in a vast range of environments. They have a limited operational life, due to the finite nature of battery power, and the risks of their operational environments. They form networks via low-powered radio communication which is very unreliable, hard to model, and consumes a great quantity of battery power. All of these constraints combine to make WSN very unreliable when compared to desktop PCs. Failure of one or more nodes is expected. Many of the networking and distributed system protocols used in the past, such as those used to make the Internet, or mobile phone networks, do not function well for WSNs. These protocols assume more stable environments, less frequent failure of the nodes and more reliable networks.

Previous focus of WSN systems research has been on operating systems and resource management at the node level, or on efficient service protocols such as those for data routing and information dissemination. There are several wireless sensor node operating systems which have arisen from WSN research projects, such as TinyOS [LMP+05], or are part of active research and development, such as Contiki OS [DGV04]. Both provide programming APIs and

are widely used. They both focus only on the simple and efficient management of the individual node on which they run. TinyOS and Contiki both offer network interfaces which can be used to create protocols with which to communicate with other nodes, and ultimately form a network. Few sensor node operating systems offer any management capabilities beyond the single node upon which they run. I propose an operating system over an entire WSN, a WSN operating system.

Separate lines of research have been done on the creation of efficient communication protocols to provide services to applications wishing to use the data collected from WSNs. There are data collection protocols such as CTP [GFJ+09], time synchronisation protocols such as FTSP [MKSL04], and data dissemination protocols such as Deluge [HC04a]. Each one of these protocols has been shown to operate reliably and efficiently on their own. Some, such as Deluge, have been used to create WSN management systems. The problem is that very little work has been done on how these protocols affect one another, and how their combination affects the performance of the network as a whole. Up to now, these protocols have been used to build WSN systems by simply adding all of the protocols needed to provide the required services. No higher-level system considerations or effects have been considered.

A limited body of work has been done on how to create and manage a WSN as an entire system. The system tools proposed tend to focus on a certain area like node failures, reprogramming the nodes, or offering data collection abstractions. The Nucleus system [DHJ+06], [AR05] focuses on query and command dissemination, and on network reprogramming. The Impala [LSZM04] system was deployed on mobile nodes, and used to collect data and disseminate code updates in a network with very unreliable communication links. TASK [BGH+] used a database abstraction to aid the way an application would access its data. In all cases little effort was made to collectively organise the network or manage its shared resources, such as the communication medium.

The current state of the art WSN management systems use individual protocols to offer specific services. This approach has several problems when used on resource constrained sensor nodes such as the MicaZ or TelosB motes [PSC05]. The use of multiple protocols can cause interference

between the protocols. Another problem is that simply adding more protocols to add more services increases communication overhead. What is required is the ability to provide more services with less communication to avoid inter-protocol interference and conserve energy.

I am not the only researcher to notice this phenomenon. Periodic heavy communication required by a code dissemination protocol called Deluge was shown to starve the Mint-Route [WTC03] data collection protocol. Deluge prevented Mint-Route from broadcasting the beacons it needed to enable data forwarding [HC04a, LBV06]. This caused the Mint-Route collection protocol to fail, delivering only 2% of the data sampled by the sensors. We have also observed the same problem in our early experiments, where high data collection rates cause data dissemination to fail. This highlights the problem that two service protocols may interfere with each others' communication, both through radio interference, and by starving each other on the sensor node. The interference can cause one of the protocols to fail.

The focus of this thesis is to create a set of tools with which to create a robust WSN distributed operating system which deals with the WSN management problem in an efficient and effective way while remaining suitable to the harsh operating conditions of environmental monitoring.

1.3 Approach: WSN management Using Bio-Inspired Algorithms.

The WSN management systems I mentioned above are predominately tools to monitor the nodes, or disseminate information to all of the nodes. They tend to take a centralised approach towards control, i.e. someone is controlling the nodes, and issues commands to them. In this thesis I propose the use of bio-inspired algorithms to let the nodes manage themselves, based on parameters set by an application or user. This is a bottom-up control approach where the network will continue to function and can adapt to change in the absence of central control, can deal with an increase in network size without requiring more control resources, is robust to the failure of any single node, and only requires the exchange of local information to function.

I define the term bio-inspired algorithm as an algorithm based on a mathematical model of the behaviour of a biological system [OZ06]. These models tend to capture the bottom-up, decentralised form of self-organisation observed in organisms such as flocking birds, ant colonies, or neurons. I recognise in this thesis that this self-organisation emerges as a result of a form of computation referred to as biological computation [Mit09]. This form of computation takes a particular bit of state on each of the nodes as input, and changes the value of that state on each of the nodes as output. The nodes change their state based on the states of their local neighbours. My goal is to enable the self-management of WSN as a result of biological computation. I show that this is an efficient way to create a WSN operating system which will allow WSNs to self-manage in a robust, scalable, and communication efficient way.

There are several different bio-inspired algorithms. Some are based on the foraging of ants, and are used as heuristics to find the solution to hard routing problems such as the Travelling Salesman Problem. Another branch of bio-inspired algorithms is based on a model of evolution, and is called evolutionary computation (or genetic algorithms) and is used to find near-optimal algorithms. I focus on bio-inspired algorithms developed from the observations of organisms that use swarm-intelligence such as flashing fireflies in South-Asia, flocking birds, and the way epidemics or rumours are spread in human populations.

Bio-inspired protocols work through the inter-communication of individual nodes. Each node communicates with and adjusts itself based on the received states of its local neighbours. Through these local interactions, global patterns emerge. These patterns are observable if you are able to see the state of all of the nodes at once. A good example of this is the synchronised flashing of fireflies. Each firefly flashes its light in accordance with the flashes of its neighbours. The emergent effect to an outside observer is that all of the fireflies flash in synchrony. If that observer were an application, then this emergent effect could be used as a timer.

I use bio-inspired swarm-intelligence protocols which use biological computation to produce an emergent result. This result is used for practical purposes, such as synchronisation, or information dissemination. These can be exported as services to the network users. The protocols comprise an operating system which provides new abstractions to users. My operating

system enables the sensor nodes to form a network where the intercommunication between the sensor nodes performs distributed computation. The result of this computation is the services which are supplied to the upper layer. The services which I will present as examples in this thesis are: time synchronisation, and parameter dissemination.

One of the key insights I exploit is that all of the bio-inspired swarm intelligence algorithms which are explored use information in a similar way. This allows me to intelligently combine the communication overhead of several different services without affecting their performance. I distinguish intelligent combination from simple combination because both data and processing of the data can be combined. The combination provides a very efficient use of the network channel capacity.

1.4 Hypothesis

The hypothesis of this work is that biological computation realized through bio-inspired protocols can be used to enable a system of low-resource devices like wireless sensor nodes to self-organise and self-manage their resources such as energy or communication bandwidth in an efficient, and in some aspects optimal, way. I see the WSN management problem as the embodiment of the problem at the core of our hypothesis. The WSN operating system tools that I present in this thesis provide evidence in support of my hypothesis.

1.5 Challenges

The WSN management problem is that system information from each of the nodes is needed to be able to manage them. The process of getting that information to a central place to process it, and then returning the result to the nodes requires communication. Radio communication used by WSN nodes is the largest consumer of the finite energy resources contained within a WSN node's battery. To solve this problem we need to find a way to move and process WSN system information in an energy-efficient or low-communication overhead way. This is so that the

nodes can be organised into a coherent network capable of providing services to an application or end user. We show a way to address this problem using biological computation through bio-inspired swarm intelligence algorithms. There are various challenges to this approach:

1. Communication resources are limited, we need to be able to provide these services in such a way that one service does not inhibit or interfere with the functioning of the other. To do this we need to explore different approaches to make this possible.

2. Bio-inspired algorithms are communication intensive, and can be so because animals can eat and create energy. WSNs are resource constrained, and have to save energy/communications, So we need to approximate the desirable aspects of a bio-inspired protocol in a more energy efficient way.

3. Bio-inspired algorithms tend to offer a single service. We need to find a way to make them more flexible so that many services can be offered to an application in a way that is resource efficient and dependable.

The reasons that I have chosen these challenges is because bio-inspired algorithms hold the possibility of performing large-scale distributed computation. This can make WSNs behave like large distributed computers in their own right. This fact may open up new possibilities and applications in computing, as well as allow the exploration of new computing paradigms.

1.6 Contributions

The main contribution of this thesis is that I add to the discussion about the use of biological computation to enable distributed management of WSNs in a low-communication, energy-efficient way on low-resource sensor nodes. My practical contributions serve as examples of how using the global results of distributed computations on all of the wireless sensors can be used to manage the network. The practical contributions I make in this thesis are:

1. The design and implementation of a cross-layer scheduler which determines which protocol an individual node should use, and which node should gain access to the communication medium

first. This is done in a decentralised way, and provides throughput optimality.

2. A lightweight and efficient decentralised global synchronisation algorithm with a global time-stamp.

3. A communication efficient set of services based on epidemic algorithms and exported to a user which have a low communication overhead whilst maintaining the desirable properties of bio-inspired algorithms.

1.7 Structure of Thesis

This thesis is organised as follows: in chapter two I give some background on WSN management systems and operating systems. In chapter three I present the background to my proposed solutions to the WSN system problems. Bio-inspired algorithms and biological computation are discussed to clarify what I mean when we say that I am enabling and using distributed computation. I give my first solution using bio-inspired algorithms in chapter four. I present a decentralised scheduler which uses only local information to schedule both protocols at the node, and nodes in the network. In chapter five I describe a decentralised epidemic-based synchronisation protocol which enables the nodes to synchronise with one-another. I show in chapter six that my epidemic approach to node self-management can be extended to include other services such as information dissemination with little additional communication overhead. Finally, in chapter seven I offer a summary of the ideas presented in this thesis, and some possible future directions.

1.8 Publications

The following is as list of the publication relating to the work done in this thesis.

- M. Breza, R. Anthony, and J. McCann. Scalable and efficient sensor network self-configuration in bioans. *Proceedings of the First IEEE International Conference on Self-*

Adaptive and Self-Organizing Systems SASO 2007, July 2007

- M. Breza, R. Anthony, and J. McCann. Quality-of-context driven autonomicity. *Proceedings of the Second International Workshop on Engineering Emergence in Decentralised Autonomic Systems EEDAS 2007*, 2007

- M. Breza and J.A. McCann. Lessons in implementing bio-inspired algorithms on wireless sensor networks. In *Adaptive Hardware and Systems, 2008. AHS'08. NASA/ESA Conference*, pages 271–276. IEEE, 2008

- M. Breza, P. Martins, J.A. McCann, E. Spyrou, P. Yadav, and S. Yang. Simple solutions for the second decade of wireless sensor networking. In *Proceedings of the 2010 ACM-BCS Visions of Computer Science Conference*, pages 7–17. British Computer Society, 2010

- M. Breza, S. Yang, and J. McCann. Multi-protocol scheduling for service provision in WSN. In *SENSORNETS 2013 - Proceedings of the 2nd International Conference on Sensor Networks*. SciTePress, 2013

Chapter 2

Background

2.1 Introduction

To transform a group of WSN nodes into a WSN system able to meet the needs of an environmental application, shared information is the most important thing needed. By system, I refer to a collection of WSN nodes which work together to provide services which no individual sensor node could provide on its own. We also take the view in this thesis that a WSN system consists of just low power sensor nodes, and does not rely on any larger, more resource rich nodes. This keeps system costs low and accounts for the fact that WSN tend to operate in very harsh conditions where failure is common. Therefore, it is unwise to have any single points of failure.

In order to form a coherent system the WSN nodes need to agree upon what to sample, when to sample, how long to sample etc. They also need some degree of organisation if they are going to communicate their samples to a base location for storage, display or further processing. Some of this information can be programmed into the nodes before they are deployed. This assumes that the requirements or the operating environment will never change. Other things, such as synchronisation, cannot be pre-programmed onto the current generation of WSN nodes (more expensive nodes can use GPS, but they are expensive, and have short operational lifespans due to the high energy consumption required by current GPS chips). In order to turn current WSN

nodes into coherent systems, information needs to be shared by all of the nodes.

The system information required by WSNs to enable them to function as a coherent system exists at a level below that of an application, and above the level of an individual node's operating system. Applications may require this system level data. For instance, my example smoke monitoring application would use the system synchronisation information to ensure that the samples sent by the sensor nodes are time-stamped. That same synchronisation information would be used by the local node's operating system to set its local event timer so that it knows when its neighbours are awake. This information is relevant to all of the nodes, not just one.

The organisation and dissemination of this needed information is the role of my WSN operating system. We refer to the principle responsibility of a WSN operating system as WSN management. The fundamental problem I face is that the information which it requires comes at a great cost. Communication is severely restricted in WSN due to the associated high energy cost, and the small amount of energy which a sensor node will have (embodied in the battery). The CC2420 radio chip used by the MicaZ uses approximately 17.4mA at full power to send, and receive. At that current draw the batteries would have a capacity of 2750mAh. The batteries would handle the constant load of the CC2420 for around 158 hours. Nodes only send and receive very little (by necessity and design) and battery discharge models are not simple, but, it should be clear that the power available to a sensor node is finite, and its conservation is a central design consideration. A WSN operating system needs to enable the sensor nodes to self maintain, and they need to do it with very simple algorithms in terms of the memory needed, processing required, and communication used.

Another problem inherent with WSN management is the unreliable nature of the low power radio communication used by the WSN nodes [BMZN+12]. Low-power radio communication is plagued by many different problems. The environment can cause multi-path propagation due to signal reflection, or attenuation through signal absorption. Interference can be caused by neighbour nodes in the network, or other forms of electromagnetic radiation. These factors all affect the reliability with which a WSN operating system can obtain the information it needs.

As an example I enumerate the information requirements needed by a system to support my

illustrative smoke monitoring application. These requirements include: collection of sampled data, ensuring that the data is sampled in a usable way, and the ability to configure and control the network. In order to fulfil these requirements a WSN operating system is needed to provide the required information as a service to the sensor nodes.

The first service is synchronisation to ensure that the smoke samples of each individual sensor are taken at the same time, or at least with in an acceptable window of time. This ensures that at a given point in time, we know how the smoke is distributed around the room.

We need to be able to disseminate control and configuration information to all of the sensor nodes. For instance, say that the smoke sampling frequency needed to change. We would want that change distributed to all of the sensor nodes in as little time, and with as little communication, as possible.

Another crucial service is the collection of data. Often, a WSN will be so large that one node will only have reliable communication with a very small subset of the overall WSN. This small subset, defined as all of the nodes in one-hop radio range, is referred to as its local neighbourhood. If the WSN has a single data collection point, also called a base station, then its single hop neighbourhood will also be a small subset of the total WSN. In order for the base-station to receive data, many nodes of the WSN will have to forward data sent from one hop neighbours further from the base-station, to those nearer to the base-station until the data is received by the base-station. This problem, referred to as multi-hop routing, is a very heavily studied problem in the field of WSN [WTC03, GFJ+09].

The problem which this work seeks to address is that of the type of system management performed by a distributed operating system in a wireless sensor network. Currently in the field of WSN, there are many approaches towards the provisioning of the information required by WSN applications, but there is very little work on the combination of these information services into a single system. There is also the question of whether many of these current solutions can be combined and will work concurrently on low resource sensor nodes.

I begin my discussion of WSN systems by briefly looking at the most commonly used operating

systems. This will show that operating system development has focused on the efficiency at the node level only. Next I will take a brief look at some of the MAC protocols proposed to manage the WSN communication medium. From there I will look at some of the most commonly cited protocols used to provide WSN applications with the information that they require to function as a system. I then move up a level to WSN management, and look at some of the most commonly cited management systems. Finally, I present and discuss the main problem with WSN management which I refer to as the WSN management problem.

2.2 WSN Operating Systems

In this thesis I propose an operating system that operates on all of the nodes in parallel, and abstracts all of the individual nodes into a single system which provides services which are comprised of information from all of the nodes. The usual notion of an operating system (OS), such as that used on smart-phone or desktop PC, is of a layer of software between the hardware and the users. This system abstracts the underlying hardware and provides a relatively simple set of abstractions realised through commands to a user or application. This makes the underlying hardware easier to use. At the same time the operating system manages the resources of the hardware. For example, it determines which program has access to the CPU, for how long, and the organisation and access to the files and memory. The services provided to the user through the OS commands are given with the best performance possible. An example of this is the way an operating system manages programs wishing to execute on the CPU. A scheduling policy is use by the operating system to ensure that all programs get a share of the CPU so that they may progress. It gives some processes, such as the user interface, priority so that the user feels that the computer is responding quickly.

WSN operating systems are much simpler that their PC counterparts. There are several operating systems developed for Wireless sensor nodes such as TinyOS [LMP+05], Contiki [DGV04], LiteOS [CASH08], Mantis [BCD+05], and SOS [HKS+05]. Each one is similar in that they focus mainly on two problems, management of resources on the individual nodes, and providing a

programming interface for the node which use the limited resources in an efficient way. They all use C either in pure form or a modification to C. They are used to program wireless sensor nodes by providing both the operating system to run the system, and the application. The resulting executable file is then uploaded to sensor nodes, and when they are turned on they execute their applications.

All of the operating systems except for one operate only on an individual node. In this thesis I propose an operating system that forms all of the individual nodes into a single system which provides services from all of the nodes together to a user. This type of operating system is important because the sensors need to be organised in order to be used by an application, and the overhead of the system organisation coupled with the requirements of an application can overwhelm the capacity (communication, energy, or memory) of the individual nodes. A WSN operating system which works on top of the node operating system is a requirement for the deployment of WSN applications, and the design needs to consider the constraints of the nodes on their own and when they work together.

2.2.1 TinyOS

The best known and most commonly used WSN operating system is TinyOS. Programming is done with a component based version of C called nesC. It uses an event based process model and is single threaded. Communication primitives are provided via an active messaging interface. TinyOS has a large library of extensions, including several communication protocols to provide various services to applications.

TinyOS programs are organised as components. Components are like very strictly encapsulated objects and are an abstraction based on electronic components and the way that they are wired together. Each component needs to have an interface, and one component may use another only by 'wiring' to it. An interface exports either commands which the wired components may call, or events, which wired components must handle. The communication expressed by the wiring metaphor, i.e. the communication from one component to another, is typed. Commands

can either return typed values directly, or through an event. Event handling is used instead of blocking for commands whose completion time is indeterminate.

TinyOS uses an event based process model, similar to the process model used in GUI development. When a command is called whose return time is indeterminate, the result will be returned by an event. An example of this is radio communication. If the network is crowded, then the radio may have to wait until it can get a free channel. Instead of having a send command block the system while waiting, the send command returns. Later, the result of the send is returned via an event when it has occurred. This model allows TinyOS to use a single thread of execution model, with two classes of processes.

The scheduler has two types of processes. Low priority, potentially long running processes called tasks, and high priority processes called events. Events preempt tasks, and will be handled when they occur. Multiple events will be put into a first come first serve queue. The idea is that events should be handled quickly to not tie-up the system. Longer processes can be executed as tasks, which will not tie-up the system.

The default TinyOS radio stack uses a messaging abstraction called Active Messages. This provides an individual interface to each protocol which needs to communicate over the radio, along with an Active Message ID for each message type. Each message type is given a buffer of one message, and all protocols are serviced in a Round-Robin fashion to prevent protocol starvation.

There is a large library of protocols to enable and support different functionality in TinyOS. There are multi-hop protocols such as the Collection Tree Protocol (CTP), and time synchronisation protocols like the Flooding Time Synchronisation protocol (FTSP). Together these protocols can be used to create WSN applications to fulfil a users application requirements.

2.2.2 Contiki

Another popular WSN node operating system is Contiki. It is written in pure C, and provides a library of macros and functions which can be used to develop an operating system to run on

a WSN node. Contiki uses a type of multi-threaded process model referred to as proto-threads.

Proto-Threads (or PThreads) are lightweight threads that do not maintain a stack. Global variables need to be used in order for information to be saved when there is a context switch. To manage the threads, Contiki has a Unix-like multi-process scheduler, and also uses events to handle non-deterministic process completion.

Contiki has a modular network stack referred to as the Rime stack, and has many different networking models readily available, from simple broadcast, to multi-hop communication. Many of the same service protocols implemented for TinyOS can also be found implemented on the Rime networking stack, such as CTP and FTSP.

2.2.3 LiteOS

LiteOS provides a Unix like set of commands and a shell environment with which to interact with the WSN network. To do this it focuses on a higher level than the other two operating systems mentioned, and tries to work at both a node level, and at a network level. It has a centralised network architecture with one node acting as basestation, which sends information and requests to all of the nodes of a network, and it uses a tree-based routing protocol to collect all of the data and responses from the nodes.

Like Contiki it allows the use of lightweight threads and uses events. It is programmed in pure C, and also includes several service protocols. LiteOS is most notable as the only operating system which views the system at two levels, both the node level, and the network level.

2.2.4 WSN OS Conclusions

The common thing among most of the current WSN operating systems is that they all relate only to the functioning of one node. LiteOS is a notable exception in that it offers a WSN system view by allowing a user to compile a basestation, and includes client functionality on the sensor nodes. This allows the use of various Unix commands on a PC tethered to the

basestation, and treats the network like a Unix system. The view of the network is highly centralised, and all communication between the nodes must be routed through the basestation. This system view is prone to scalability problems because as the population of nodes increases, the amount of communication flowing to and from the basestation will also increase. This will tax the energy usage of the nodes near the basestation who are used to forward information to the basestation, and will lead to communication medium saturation if the population becomes too large, or the data rate to the basestation becomes too high. Aside from LiteOS, none of the current WSN operating systems focus on the notion of a system beyond the local node.

As it currently stands, the above mentioned node-level operating systems will further services to organise the individual nodes into a system usable by a WSN application. In many cases these services will be the same, like information dissemination to unify the information on all of the nodes in a network, time synchronisation, so that all of the nodes can function at the same time or management of the communication medium, so that all of the nodes can communicate as efficiently as possible, and with a minimum of interference. These inter-node organisational issues which are generally not addressed by the node level operating system is the focus of the WSN system level operating system proposed in this thesis. A WSN operating system would organise a group of individual nodes in an efficient and scalable way while providing services from the WSN as a whole to an application.

2.3 WSN MAC Protocols

A great deal of effort has gone into the development of MAC protocols for WSNs [BDWL10, KM07]. This is because WSNs are truly distributed systems, in which communication is a core system component. As stated in the introduction, one sensor node can not do much, but many sensor nodes working together can reveal hitherto unexplored areas of distributed phenomenon. A curious result of the great amount of effort spent on WSN MAC layers has been a plethora of proposals.

WSN MAC layers can be classified into one of two general approaches. The first approach is

the group of MAC protocols which use contention based schemes similar to CSMA. Examples of these are the default TinyOS B-MAC [PHC04], SMAC [YHE02], X-MAC [BYAH06], and WiseMAC [EHD04]. These schemes listen to the radio medium for a random period of time before transmitting. In other words, contention based protocols try their luck when they want to transmit.

The other major approach is the group of WSN MAC protocols which use schedule-based schemes similar to TDMA. Members of this group include: PEDAMACS [EV06], FLAMA [RGLAO05], TRAMA [ROGLA06], and APRMAC [QMG10]. Time is organised into time-slots, and then schedule the transmission of each sensor node to occur during a unique time slot to avoid communication collision. The nodes in schedule-based protocols are told when to transmit. There exists, of course, MAC protocols that combine the two approaches into a hybrid, such as ZMAC [RWA+08] and Crankshaft [HL07]. The many varied MAC layers proposed point to a problem in the way that the MAC layer in viewed in WSN.

2.3.1 Contention Based Protocols

BMAC is the seminal example of a contention based MAC protocol of WSN. It is the MAC protocol used in all of the work in this thesis. Sensor nodes needing to transmit wait for a random period of time before listening to the communication medium to do a clear channel assessment (CCA). If the channel is clear, the sensor node transmits its packet. If the channel is not clear, it waits for another random period before doing another CCA. Before transmitting data a preamble is sent that is detected by other nodes sampling the medium while waiting to send. BMAC introduces an outlier detection technique to make accurate CCA assessments. The preamble increases transmit time therefore wasting energy. In heavy traffic conditions or dense networks, BMAC suffers long latencies, and therefore has poor throughput, due to long back-off times.

The X-MAC protocol reduces the preamble time by strobing the preamble so that a receiver can respond with an acknowledgement message, thereby reducing the length of the preamble.

This scheme only reduces the preamble for unicast communication, and does not provide any reduction for broadcast communication.

A different approach to preamble shortening is taken by WiseMAC. WiseMAC assumes duty cycled nodes (where the nodes sleep for period of time to save energy). Each node keeps track of the wake-up times of its neighbour. The wake-up time information is combined with MAC protocol acknowledgements. Using this information a sender can know the wakeup time of a receiver, and then use a short preamble for communication. This scheme also does not allow short preambles for broadcast messages.

Extending the idea of organising communication around duty-cycling, SMAC uses synchronisation packets to synchronise sensor node wake-times. After initialisation, all of the nodes in the network are synchronised. The wake-times are used for communication. The wake-times are divided into two parts. The first part is for synchronisation and control messages, the second part is for the exchange of data. By organising communication around the duty-cycle schedule, SMAC begins to approximate a scheduled MAC protocol. SMAC suffers from high data rate applications or dense networks when the population of senders and the rate of sending become larger than the communication window permits.

2.3.2 Schedule Based Protocols

Power Efficient and Delay Aware Medium Access Control protocol for Sensor Networks (PEDAMACS) has a central sink node which uses information about the topology of the sensor nodes and the traffic pattern on each communication link to devise a collision-free schedule based on a graph-colouring algorithm. This MAC protocol only supports the all to one communication pattern, and does not provide well for broadcast communication.

The FLow-Aware Medium Access (FLAMA) and TRaffic-Adaptive Medium Access (TRAMA) protocols both use a distributed scheduling approach to organise access to the communication medium. Both protocols assume that the long-term communication patterns of the nodes will be stable. Each node broadcasts its traffic-flow and a list of of its one-hop and two-hop

neighbours. This information is used by a distributed hash-function on each node to determine which time-slot it will send in. The resulting schedule avoids interference from hidden terminals because it uses two-hop neighbour information. FLAMA differs from TRAMA by reducing the frequency of the information communicated between nodes to only do so when a change occurs. Both of these protocols sacrifice energy efficiency because they use a high degree of inter-node communication to determine their schedules.

The Aerial Platform based Routing and Medium Access Control protocol (APRMAC) is a scheduled protocol that uses a centralised controller node located in an aerial platform above the WSN. It requires that the nodes send their one-hop neighbour information to the aerial central controller. The controller uses the one-hop neighbourhood data of each node to determine the shortest path route from each node to its data sink. The controller uses the routes to find a collision free communication schedule for each of the sensor nodes. This approach is interesting because it removes the control from the nodes, and reduces the inter-node communication needed by protocols like FLAMA. This approach is, however, susceptible to the failure of the single controller node, and therefore may not be robust enough for some outdoor or long term applications.

2.3.3 Hybrid Protocols

There has also been work done on MAC protocols which combine the contention-based approach and the scheduled approach. Zebra MAC (ZMAC) uses a distributed slot allocation algorithm to create a TDMA-like transmission schedule assigning a slot to each sensor node. Under low communication conditions, this schedule is sufficient, and each node can communicate all of its data in its slot. If a sensor finds that it is unable to send all of its data during its allocated time slot, it attempts to use an unallocated time slot. To use an unallocated time slot, a node starts a CSMA-like random back-off timer at the beginning of the unallocated slot it wishes to use. When the back-off period is over the node uses the unallocated slot, provided that it is still unused. ZMAC's distributed slot allocation algorithm requires a great deal of communication. The schedules it produces get altered in times of heavy traffic by nodes claiming unused slots.

This causes schedule skew and requires that the distributed slot allocation algorithm be re-run. Re-running that algorithm during a time of heavy communication compounds the network congestion.

A MAC protocol called Crankshaft divides time into frames, and further divides frames into slots. The slots at the beginning of a frame are unicast slots, and belong to one or more sensor nodes based on the value of their MAC addresses. Communication within a slot is handled by using a random back-off time and channel sampling similar to CSMA. The latter slots of a frame are used for broadcast traffic. The crankshaft protocol functions well in dense WSN deployments, but requires too much overhead to be used in WSN with sporadic or bursty traffic patterns.

2.3.4 WSN MAC Protocols Conclusion

Many different MAC protocols have been proposed, most heavily tailored to specific network conditions. Work presented in [BDWL10] suggests that the different approaches to MAC layer scheduling are appropriate for different network communication loads. A simulated 1000 node network with Poisson distributed traffic was used to evaluate the performance of different MAC layers at different data rates. The simulation results suggest that the contention based MAC protocols work better at lower data rates, and the scheduled MAC protocols work better at higher data rates.

It is my view that the communication medium is an integral part of a WSN system, and needs to be managed globally by a WSN operating system which exists at a level above all of the individual nodes, like a desktop operating system manages the CPU and all of the components of a desktop computer. The fact that there has been so much work done on MAC layers shows that the problem of network medium access and use is a core problem for WSN system formation, and one that should be handled from a WSN system point of view, above the level of individual nodes. The control and management of the communication medium is crucial to formation of WSN systems. The current view of a MAC layer as an isolated layer is based on the OSI model [Zim80] is insufficient for WSN. This is because WSNs rely on network connections

more than desktop computers, and their network conditions are much more varied than the wired communication for which the OSI model was devised. The impracticality of the OSI model for WSN and its view of medium access and control is also observable by the amount of cross-layer optimisation that is present in WSN service protocols [LSS06].

2.4 Common WSN Service Protocols

There are several state-of-the-art protocols which are common across WSN operating systems. These protocols provide different services for different tasks, including routing, time synchronisation, and information dissemination. It is by using these protocols that a group of sensor nodes form a network and provide services as a WSN.

2.4.1 Data Routing

Routing protocols are used to collect data sensed at the node level and deliver it to an application or user. Wireless nodes have a very short communication range due to their low powered radios. They will have to forward data through neighbour nodes in order to deliver data. The most commonly used routing protocol at the moment is the Collection Tree Protocol (CTP)[GFJ+09]. This protocol builds shortest path trees from each node to a collection sink. Each hop uses a metric based on the expected packet reception ratio of a neighbour, and all of its neighbours leading to the sink. The next hop neighbour who's metric indicates the best route to the sink is used to forward its data.

CTP uses a metric referred to as the expected transmission ratio (ETX). It is calculated as the inverse of the packet reception ratio (PRR). The packet reception ratio gives the ratio of messages received by a neighbour to the number of messages sent to that neighbour. It is calculated on a per neighbour basis and gives a measure the reliability of a communication link. When a packet is sent to a neighbour, an acknowledgement (ACK) is requested. The PRR is the ratio of ACKs received per packets send. This metric give an approximation of how good

a communication link is from a sender to a receiver. The method is not perfect, because the failure to receive an ACK can mean either of two things: The link to the receiver is bad, and the packet never arrived, or the packet arrived, and the link from the receiver to the sender is bad, and the ACK was lost. The first situation is fine, and what I am measuring. The second situation will effect my estimation of the out bound link to a neighbour in a negative way. It might be that the link to a neighbour is fine. But if the reverse link is bad, then we may never know. Asymmetric link qualities are a major problem with low power radio communication.

With CTP, each node measures its outgoing links to its neighbours by using one over the PRR, or the ETX. The nodes use not only the ETX of their immediate links, but also use a summation of all of the ETX values of all of the upstream neighbours of all of its neighbour links to find the best route to the sink. Using these path summation estimates, CTP finds the best route in terms of shortest path and best link quality.

CTP is capable of finding very good data routes, and it is responsive to changes in network topology. It does take a while for a CTP network to discover its routes, and this affects the speed at which the protocol can adapt to change. Its data rate is reliable to about 2 packets a second received at the basestation (noted in both the paper cited above, and observed by the author on networks of MicaZ, TelosB, and Iris motes). The traffic CTP generates to build its routes can be rather high, and because of this it can starve other protocols. This was observed by the author, and will be discussed later in this chapter.

The Routing Protocol for Low-Powered and Lossy networks (RPL) is a draft standard for WSN routing using IPv6 [TED10, KTDH$^+$11]. Its aim is to enable individual WSN nodes to be addressable on the internet. RPL works by constructing a destination oriented directed acyclic graph (DODAG) from all of the WSN nodes in a network to one or more root nodes. It has special messages propagated by the root nodes which contain an ID number for the DODAG, and a value indicating the age of the network information. Network information, such as the ID's of root nodes and next hop neighbours will have a high rate of change as RPL is designed to function on networks of unreliable nodes. These same ID messages are used to determine the hop count from a node to its closest root. RPL uses CTP's link quality metric

ETX to determine a nodes next-hop neighbour for routing information to its closest root. It also uses an epidemic dissemination algorithm called Trickle to ensure that every node has the most up-to-date network information. RPL aims to enable both a many-to-one traffic pattern for data collection, as well as a one-to-many communication for information dissemination to all of the nodes in a WSN.

The problems with RPL are the same as with CTP, except that this protocol maintains network state, and aims to keep that state consistent among all of the nodes in the network. RPL's state consistency mechanism can add communication overhead and, as I will demonstrate through experimental results later in this thesis, can cause inter-protocol interference.

Another notable and interesting new routing approach is Backpressure Routing (BCP) [MSKG10]. This protocol uses node queue lengths to find throughput optimal routes through a network to deliver information to a base-station. Every sensor node keeps a queue of the messages it wishes to send to the base-station. Nodes closer to the basestation will have lower queue lengths. The base-station itself will have a queue length of zero. This means that a gradient will exist from any sending node to the base-station. Each node makes a local decision to forward its packets to its neighbour with a shorter queue length. Formal analysis of the routes chosen show that they are throughput optimal.

BCP is shown in published experimental results to respond better than CTP at high rates of data. Because it needs queues to function, it can have a long latency for data packets. BCP uses a first in first out (FIFO) queue behaviour in order to reduce message latency. This behaviour means that some messages may never get delivered. BCP handles this by using virtual queues, which pad out the lower positions of the FIFO queue (those that would never get serviced) with null packets to ensure that real packets are high enough to always get sent.

Two problems exist for BCP. The first is that the queue it requires to function require a great deal of memory. Wireless sensor nodes are usually very memory constrained. This makes large data structures like message queues impractical. The MicaZ node only has 4Kbytes memory for its data structures. This means that the message queues need to be short. There is no memory protection on the Atmel Atmege128L used by the MicaZ. This means that allocating

too much memory will cause unpredictable and hard to debug errors. The other problem with BCP is that it may chose long routes, and those routes may contain loops. The author has observed queue lengths up to twice the minimum hop depth of a testbed using BCP.

2.4.2 Time Synchronisation

Synchronisation refers to processes or events occurring at the same time. It comes from the Greek word synchronos which means the equivalent time (syn - equivalent to , chron - time, os - adjectival suffix). In this thesis I recognise two different types of synchronisation. The first form of synchronisation is referred to a global synchronisation, and implies the existence of a global time-stamp which all of the nodes can produce at the same time. It is analogous with every one in a room having the same time on their wrist watches. The second type of synchronisation is called event synchronisation where every node shares an event frequency with their neighbours. Event synchronisation means that all the nodes align the phases of their event frequencies, and all perform the same event at the same time. This is analogous with everyone in a room clapping at the same time, regardless of the time on their wrist watches. The difference between global and event synchronisation is time values on their clocks. Globally synchronisation means the same time on every clock. Event synchronisation means a different time on every clock. In both cases, an event will occur at the same time.

An example of a commonly used synchronisation protocol for global synchronisation is the Flooding Time Synchronisation Protocol (FTSP)[MKSL04]. It assigns an arbitrary root (the node with the lowest ID), and all nodes receive and buffer synchronisation messages from that root. When enough samples have been gathered (the default is 4), then the nodes calculate their individual offset from the clock of the root, and their skew from that clock (their rate of drift from the root's clock). This protocol works in a multi-hop network, and claims to be able to provide timestamps with an accuracy of 4 microseconds. FTSP provides global synchronisation and ensures that every node can produce the same time-stamp. The time-stamps is an arbitrary value, taken from the root, but will be the same for each node at the same time.

The main problem with FTSP is that it is reliant on an arbitrary root node. This means that

if the root node were to fail, then a period of instability would ensue while the new root node was determined. The other problem with FTSP is that it requires its own communication, and can fail when the communication demands of its neighbours becomes too high.

2.4.3 Information Dissemination

Information dissemination is the process whereby information is given to every member of a group. It is important in computer systems because they are information processing devices. The information itself can be something large like the operating system of a desktop computer, or something small like the number of minutes until the screen saver on the same desktop computer turns off the screen to save energy. In distributed computer systems, like environmental monitoring WSNs, it is important that all nodes have the same information so that the requirements of the application can be met. This could be as simple as ensuring that all nodes sample their sensors at the same rate, or that all nodes use the same protocol so that they may intercommunicate.

Information dissemination is an especially interesting problem in WSN. This is because the systems can be very large, so that the dissemination protocol has to be able to scale. It is conceivable that an environmental monitoring network tracking soil moisture in a large farm may scale to thousands of nodes. Another challenge is the unreliable nature of WSN communication. Low power radios are very prone to message loss, and environmental factors such as growing foliage or moving bodies can greatly affect the quality of a nodes communication. The final, and the largest challenge, is the energy constraint problem faced by WSN. The very communication required by nodes to disseminate information is also the largest consumer of energy.

The other reason why an automated information dissemination system is required is that it would be difficult and time consuming to have to manually change the information of a large WSN. A potentially large amount of time and effort would be required to collect every sensor node, attach some sort of communication cable, and reprogram the node by hand. In the laboratory in which I work, deployments of more than twenty nodes would take me almost half

an hour to collect, reprogram, and re-deploy when running experiments. Another complicating factor is the placement of the sensor nodes. One of my WSN deployments was on the top of a Victorian bell tower with 324 steps to reach the deployment area. In this sort of location, information dissemination is very difficult to do manually.

Two example dissemination protocols in WSNs are Drip [L+03] and the DHV [DBFP09] protocols. Both protocols are epidemic protocols where each node periodically advertises the version of its current data. The period of the advertisements change depending upon the existence of new data. When there is no new data, then the advertisement period is long, saving energy. When a node detects the presence of new data, it reduces its advertisement period to a much smaller one, in order to speed up dissemination of the new information. Drip is used by the Deluge protocol mentioned above to detect when there is a new code image. DHV is a variant of Drip which uses hashes of the data to identify the existence of new data with a smaller data overhead than Drip.

The problems with these dissemination protocols is that they do not combine well with other protocols. Their steady state communication requirements are low when there is no new data to be disseminated. When new data is introduced, then the dissemination protocols tend to use all of the bandwidth available to transfer the new information across the network. This bursty traffic pattern can cause other protocols in the network to fail. An example of this is the failure of the Mint Route protocol when a protocol using Drip (called Deluge, and discussed later in this thesis) were used on the same network [HC04a, LBV06]. This caused the Mint-Route collection protocol to fail, delivering only 2% of the data sampled by the sensors.

RPL, mentioned above, also uses Drip. Each node sends periodic advertisements to its neighbours of its routing information. If new data is discovered, the advertisement period is shortened until the new data has been disseminated, and all of the nodes are consistent. The ability of this routing protocol to co-exist with other service protocols in actual deployments remains to be seen.

2.4.4 Service Protocol Conclusion

Implementations of these protocols can be found in both TinyOS and Contiki, the two most commonly used WSN operating systems. Often, these protocols will be combined together to provides services to WSN applications. On their own these protocols provide services to applications or users. In isolation they tend to function well and and with efficient communication use. The problem is that in order for an application to function, it often needs many different services. For instance, my smoke monitoring application could be produced using FTSP to synchronise the nodes, Drip to disseminate commands, and CTP to collect the sensed data. The question then is, what happens when all of these protocols function concurrently? Will they all be able to function properly, or will they interfere with one another. We return to this question shortly.

2.5 WSN Management

Next, I are going to present WSNs from a system point of view, and focus on the WSN management function that I want a distributed WSN operating system to perform. Our definition of system is a group of sensor nodes working together to provide spatial data in time. The key point of my system definition is that many nodes work together, and that they fulfil their purpose only through working together. One sensor node would not be able to measure the smoke moving around a large room. It would only give one data point, and alone, this would be useless. When working together, the data that can be collected by a group of sensor nodes can give new insight into difficult to model phenomenon, like the dynamics of smoke in a crowded room.

It is important to note that when I refer to a system, I refer to the sensor nodes only, and larger, more resource rich computers as being external to the system. They may be clients or base-stations collecting data from the system of nodes, or controllers issuing commands to the nodes. But they are outside of the system in that they can connect to the system at any, arbitrary point, and still find the same functionality. The system also does not required the

larger nodes to function. In the absence of a base-station, the nodes will still collect data. Only without an audience.

This discussion is based on overviews of WSN management systems given in [LDCO06a], and expanded by the functional requirements and constraints discussed in [TC], [MLM+05], and [HMCP04]. We will look at the functional requirements of systems that perform WSN management have and the constraints which they have to work within.

2.5.1 WSN Management Constraints and Requirements

The most important constraint of WSN management is energy. Energy is the main limiting factor in the operation of a WSN and communication is the largest consumer [CSR04]. Wireless sensor nodes are battery powered making energy a finite resource. The rate of energy use in a WSN determines the operation lifetime. WSN management systems must be able deliver their functionality with as little communication as possible. This means that the communication protocols used must be very efficient.

Both limited energy and difficult operational environments means that failure is common in WSNs. This gives us the requirement of robustness and fault tolerance. The management system must be resilient to the inevitable network instability that is inherent in WSNs [MOH04]. Examples have been given in the WSN literature of sensor nodes placed inside of glaciers [MOH04] and strapped to Zebras [LSZM04]. These sorts of environments are very harsh for any computer system, and failure will be the norm.

Related to fault-tolerance is the requirement of adaptability. The management system should be able to adapt to network variability. The failure of nodes is common in WSN systems, as is temporary partitioning of the WSN due to communication interference. The management system and its protocols need to be able to adapt to changing network condition and continue to function in the event of node or communication link failure.

Sensor nodes tend to have low memory resources. This means that the WSN management system needs to have as low a memory footprint as possible. This refers to both executable

code size, and any data structures used to record state.

Scalability is another requirement that emerges from the inherently distributed nature of WSNs. At the moment, the largest WSN deployment that the authors are aware of is just over 1000 nodes [AR05]. The vision of sensor networks is that they can encompass vast networks of sensor nodes possibly into the thousands or tens of thousands. WSN management systems need to be able to scale to large networks to be useful in the perceived future applications of WSN.

The last requirement is that the WSN management system should be as separate as possible from the application it is supporting [TC]. This is so that the management system does not impede or degrade the results of the WSN. Given the low resources on a WSN node, this is a real possibility. This requirement emphasises the need for a general approach to WSN management, and is important for the robustness of the management system to possible application errors or problems. Below is a summary of the non-functional requirements:

- energy efficiency.
- small memory footprint.
- fault tolerance and robustness.
- adaptability due to node failure or network variation.
- scalability.
- separate from application, non-dependant on application.

2.6 WSN Management Systems

Now I examine some WSN management systems and discuss how these management systems work. We evaluate these management systems based on the criteria given above. First I look at management systems which have been used, and in some cases refined in deployments. Then I look at a couple of other management systems which have only been suggested in literature.

2.6.1 Nucleus

Nucleus was used in two deployments [DHJ+06], [AR05]. It is composed of the Sensor Network Management System (SNMS) and Deluge. SNMS is used to monitor the health of the network and send commands to the sensor nodes. Deluge disseminates code updates. Both protocols use the Drip epidemic dissemination protocol to disseminate information to all of the sensor nodes in a WSN.

Deluge

Deluge [HC04b] is used for network reprogramming, and is built on top of Drip. Drip is an epidemic protocol used to inform the nodes of the network when there is new code available. Deluge provides the mechanism to disseminate large data objects. It adds a three step (advertisement-request-data) protocol, and puts the nodes in one of three states: maintain, request, or transmit.

In the maintain state the node uses the Drip protocol. If a node hears a broadcast of a code version newer than its own it listens for a short period of time. If in that time the node hears a packet that it needs, it remains silent for another time period, hoping to hear more packets it needs. If not, it transitions to the request state.

The request state is where a node actively requests the packets it needs to receive a page of data (default is 1024 bytes). A node does not have to be in the request state to receive packets, just to request them. The requests use selective NACK, with a bit vector indicating the packets needed. Requests are sent with a random back off interval added to reduce network congestion. If no packets are received after a given time period, then the node will request again. If several requests do not yield a sufficient reception rate, then the node will transition back to the maintain state. If the page is received all in one piece, the node transitions back to the maintain state.

The transmit state is where a node transmits the packets for the pages that have been requested. It aggregates all of the requests it has heard, and transmits all of the packets in round robin

order. Once a node has sent all of the requested packets, it transitions back to the maintain state.

Deluge breaks its data objects into pages, which are comprised of packets. Breaking up objects into pages allows requesters to update only certain pages, and to propagate data objects using spatial multiplexing. This means that instead of a node waiting for the transfer of an entire object before offering it to another node, it can start sharing pages as soon as it receives a page. This means that propagation time across a network equals the time it takes for one page to cross the network, plus the time to clear the pipeline the page took to arrive. Not using pages would mean that every node would have to wait to download the entire data object before offering it to another node. The time would be the time for one transfer multiplied by the number of nodes needed to cross the network.

Nucleus is the most commonly found WSN management system found in WSN deployments found in literature [SSW+, DHJ+06, AR05]. Two of its underlying protocols, SNMS and Deluge are built on top of Drip. Since Drip is an epidemic protocol, where each node communicates only with its neighbours, and each node adaptively changes the length of its listening period depending the need for updates, we can call Nucleus a decentralised WSN management system, using bio-inspired algorithms.

SNMS

SNMS [TC] is a system which allows a user to monitor the individual sensor nodes in a WSN network. The design is based on four principles: small memory footprint, create network traffic only in response to user input and none during steady state operation, be simple and robust, be as separate from the application as possible even down to having a separate network layer. SNMS allows a user to query network and sensor state, and log events in the network. The state of a nodes battery, or the recent values of its sensor readings can be obtained, and used to predict the future state of the sensor network. Thresholds can be set on the nodes, so that if the threshold is reached, then the node sends its data.

The SNMS network layer uses two different communication styles: collection and dissemination. Collection is used to collect information about the state of the network. Dissemination is used to send management commands and queries about the states of the nodes.

The SNMS collection mechanism uses a collection tree construction protocol. To follow the design principle of no network traffic during steady state operation, no tree maintenance is done, and a tree is only constructed in response to a query. A query is issued along with a tree construction messages. These are forwarded from each node at a randomly staggered time to avoid saturating the network by flooding. The messages contains a summation of all of the received signal strengths of all of the nodes through which it has passed. This constitutes a route from the node making the query to the receiving node.

On the nodes, no neighbour table is maintained. The node uses a combination of the received signal strength of the received tree construction message, along with the summation of all of the signal strengths along the path as a metric to determine which node to forward its response to. is used by each node to determine which upstream node it will route to. In this case, the neighbour with the highest signal strength is chosen as the parent node.

Received signal strength only indicates the quality of inbound link from the parent node to the local node. In order to determine if the parent node is receiving the local node's messages, acknowledgement messages are sent from the parent to the local node. A sliding window average is maintained of the acknowledgement messages received from the parent node. This average is combined with the signal strength metric, so that a better parent will be chosen if the node to parent link is poor. If the link is asymmetric, then when another tree construction message is sent, a new parent node can be chosen.

Drip is used by SNMS to issue commands and queries to the nodes it the network. The states of software components in the nodes are queried by giving each component a 2 byte integer key, and looking the key up in a schema file. A query is sent as a list of keys, and a sample period. The sample period has a random time added to it to prevent network congestion, and then is returned with the responses in the same order as the specified keys. Queries can be sent one at a time, or the requester can send queries in a loop for a continuous monitor.

SNMS also provides an event logging mechanism, which can be used for debugging. Software component authors can specify log events, which when reached, will write a log message to the RAM of the node. The log can then be read by SNMS by sending a playback command using Drip. The nodes then send the responses back over the collection tree.

Issues with Nucleus

The problems with Nucleus are that although the approach of using epidemic protocols was good, it did not go far enough. It could have provided more services using the same protocols, this could have been done by coupling the protocols together better, and therefore the multiple protocol approach took no consideration of the communication medium and its usage.

Only two services were offered by Nucleus, node monitoring and code updating. Both of these services were for the network administrators and were completely apart from the needs of any application using the network. This is not a problem on desktop PCs with large memory and energy resources, but on constrained devices such as WSN nodes, more utility needs to be gained from any code or protocol. Interestingly, both the code update announcements of Deluge and the commands queries of SNMS were disseminated using Drip. The way that Drip works is that each protocol would have its own channel, meaning that if both protocols wanted to disseminate information, they communicate separately. This would increase the communication overhead of the network, and leads us to the third criticism of Nucleus. All of the functioning of the protocols occurs at the application layer. The collection tree protocol uses link layer information, received signal strength and acknowledgements. But, there is no attempt to manage the usage of the radio by the protocols which require it, or the communication medium. As Nucleus itself is composed of three protocols, and that an application will have at least a data collection protocol, then failure to manage the radio and communication medium may cause problems.

2.6.2 Impala

Impala [LSZM04] is the WSN management system used in the ZebraNet deployment. In this deployment the nodes were mobile, attached to roaming zebras. Internally, the Impala middleware receives five kinds of events from the system layer: device, for a hardware device failure; data, to inform that a reading is ready from a sensor; packet, to signify that a packet has arrived from the network; send done, a network packet has been sent or failed to send; timer, a timer has gone off. Each of these events is received first by an event filter. This layer decides if the events need to go to Impala's Application Updater, Application Adaptor, or go directly to one of the applications sitting on top of Impala. Events are handled sequentially upon arrival, and events may not block. All network I/O is handled by another system component and handled asynchronously.

The Application Updater manages network reprogramming. The algorithm it uses to propagate code is essentially an epidemic, gossip-based protocol, in part necessitated by the fact that the nodes are mobile. A code update is released into the system by sending it to as many nodes as the mobile base station can encounter. When two nodes meet they first transfer sensor data to one another. Then the nodes 'gossip' about the version of software they have by comparing lists of the software modules and their versions. The gossipping takes place in a three stage process. First the software lists are compared. Second, each node checks its own software versions against those received. Finally, if the node needs a newer version of the software it makes a request, if it has a new version, it sends it. During this process, the host animals may move, the system might run low on power, or the communication window may come to an end, all of which would interrupt the transmission of the code update. In this case, the module being downloaded is check-pointed until the rest of the module can be found and retrieved. In this case the older module would continue to be used. Once the whole module is successfully transferred, the Application Updater installs and links the new binary.

The Application Adaptor allows a node to adapt to a given pre-defined set of states based only on local information. The Adaptor monitors a list of application parameters like recent sensor reading values, and a list of system parameters, like current battery levels. An Application

Parameter Table is used to track which application parameter relates to which application executing on the node. The nodes in ZebraNet ran four applications each. At a given interval the Application Adaptor checks all of the above parameters, and then checks some predefined switching rules, to determine if an application switch is needed. This process is performed at the end of the nodes communication window. The adaptor then checks the parameters against the Adaptation Finite State Machine to determine if any system states should be changed. An example is, the first ZebraNet deployment used two different protocols and had two types of radio. A flooding based protocol was used on a long range radio to try and locate another node for communication. The long range radio was very power hungry. If another node was found then the short range radio, with less power usage than the long range radio, was tried. This radio also used a more efficient history based protocol. So, if the system was using the long range radio/flooding protocol, and it was found that the number of neighbours in the neighbour parameter was above a certain threshold, then the Application Adaptor would change the radio/protocol to the more efficient pair.

The memory footprint of Impala was 5712 instruction bytes and 51 bytes of data memory. This is on top of the roughly 10k bytes needed for the ZebraNet firmware and two buffers of 124 bytes and 64 respectively.

Impala was designed and worked well for only one application with very specific network conditions. The network the nodes would form was not very dense, and so the communication protocols did not have to be very efficient. A core assumption was that parts of the network would be partitioned for long periods of time, as different groups of Zebras roamed in different locations. This meant that the protocols were designed to be delay tolerant. It is doubtful that the protocols would work well in a dense or fixed network environment. The main communication efficiency was from the use of different types of radio, which is not very flexible. It is hard to see how the Impala system could be generalized. In my case I look at the management of various services and protocols in a dense network.

2.6.3 TASK/TinyDB

TASK [BGH+] is the management system used in a WSN to monitor Redwood trees. TASK is built on top of TinyDB [MFHH05], and adds: duty cycling for power management, time synchronisation, epidemic query sharing, a watchdog timer to reset in event of errors, and data logging to on-board memory.

TinyDB treats a WSN as a distributed database, and allows the queries to be made over this database using a language called TinySQL. Queries can be for sensor data, network topology discovery, and system parameters of the sensor nodes. TinyDB organises the WSN as a routing tree rooted at the base-station node. The nodes use link quality to determine their parent, as well as using an overlaid semantic routing tree to reduce the need to send queries down branches of the tree that do not meet the semantic requirement.

TASK adds a reliable query sharing facility to the network. The protocol described is essentially gossip in nature. Nodes listen to other nodes data packets. If a node overhears a data packet, it checks an 8-bit query id in the data packet with a list of the query ids it is currently processing. If the node discovers that it does not have that query, it then makes a query request to the sender of the data packet. The sender will respond by broadcasting the query. This basic protocol is subject to the creation of network congestion. To prevent this the query request messages include a bitmap to indicate parts of the query that are needed, to reduce the response size. If a node hears a request for a query that it needs, it will remain silent, and receive the query response when it is sent. Nodes are also limited to one query request message per sample period.

Power management through duty cycles is also added to TinyDB. Although the version of TASK mentioned in the paper used fixed duty cycle times, it did mention that the use of adaptive duty cycles would be better. The logging added by TASK is similar to that mentioned in other WSN deployments in the literature such as the Redwoods deployment [BGH+], Pinjar Network [COKSM05] and PipeNet [SNMT07].

TASK is different to the previous two management systems because its system view is an

abstraction, and not of single nodes. However, when we peer under the bonnet, we find another gossip like protocol. Individual nodes use only local broadcast information and their local state to determine if they need to request a circulating query or not. So, even though the exported system view is an abstraction, at a lower level, TASK/TinyDB functions by individual nodes making their own decisions.

The TASK system is implemented purely in the application layer, and makes no attempt to manage the communication or radio resources. It only provides management services to a network administrator, and does not support the functioning of an application. If an application has other requirements, such as time synchronisation, then this has to be provided by another protocol. The inclusion of extra protocols may cause communication problems because of the addition of extra communication.

2.6.4 TinyCubus

TinyCubus [MLM+]is a middleware layer to aid the development and running of adaptable WSN applications. It consists of three parts: the Tiny Data Management Framework, a cross-layer communication framework, and a configuration engine.

The Tiny Data Management Framework allows the dynamic reallocation of different data management and system software components. Each component is classified by a three-dimensional tuple, each dimension relating to a parameter. The parameters are system parameters, application requirements, and optimisation parameters. To allow adaptation, the system looks at a certain component it is using. If the component is no longer optimal along all three dimensions a new component is chosen, and the system adapts. The Tiny Cross-Layer Framework allows communication across software layers, to enable optimisation.

The Tiny Configuration Engine handles code distribution. This is done using the Topology Manager, which assumes a heterogeneous network, where there are different kinds of nodes with different functional node roles. The distribution algorithm uses flooding, but constrained only to nodes with the same functional role. The algorithm also uses random back off to avoid

the broadcast storm problem [NTCS99].

TinyCubus aims for adaptation, using some local data, but the literature never clearly specifies how it will determine when to adapt. The use of a cross-layer communication optimisation was only for a protocol to optimise its own performance. It neglected the performance of several concurrent protocols sharing the same network.

2.6.5 MiLAN

Middleware Linking Applications and Networks (MiLAN) [HMCP04] is a proposed middleware which uses information from the network layer and the application layer to reconfigure the network to the needs of one or more applications. Each application gives MiLAN a graph of its possible states, and its communication requirements at each state. MiLAN also has a list of the maximum data rate a given network protocol can provide. This information is used to dynamically reconfigure the network to use the best network protocol to optimise the application's received service level and the network energy expenditure.

MiLAN proposes to do this by sitting under the application layer and choosing a network protocol most suitable for the requirements of an application, or a combination of applications sharing the same WSN. The middleware enables an application to leverage network protocols with different specialisations to achieve its communication and energy requirements. Because of this, MiLAN is not linked to any specific network protocol.

This is one of the only systems which addressed the problem of ensuring that multiple applications get the quality of service which they need to function properly. From the literature, it was unclear what the system architecture would be. The type of processing required to perform the optimisation described in the paper would be difficult to perform on a low-resource sensor node. If it were organised as a centralised management system, then it would run into problems of robustness and ability to adapt to network change.

WinMS/FlexiMAC

The WinMS WSN management system [LDCO06b] is a policy based management system, built on top of a TDMA MAC layer called FlexiMAC. The MAC layer first creates a routing tree starting from a base-station. Each node then determines its own radio transmission, reception, forwarding, and maintenance TDMA time slots starting from the lowest node id.

WinMS builds upon this MAC layer, and manages fault detection and repair, performance, accounting, and configuration. It also enables nodes which need more communication resources, because they have more important data, to borrow the time slots of other nodes with less important data to send.

WinMS is inherently centralised. It uses a central node which performs many management functions without which it would not function. FlexiMAC pushes much of the radio management down to the individual nodes, making the network robust to link quality dynamics and node failure. WinMS adds a central node which arbitrates time-slot usage among the nodes. This dependence on a central node for core network management tasks seems to go against self-reliance given to the node by FlexiMAC.

Zone-Based Fault-Tolerant Management Architecture

The Zone-Based Fault-Tolerant Management Architecture (ZFTMA) [KMAB10] is a hierarchical, cluster based fault detection mechanism. It divides the network into four zones and randomly assigns each a cluster head. The cluster head makes a list of each node in its zone, and monitors the transmissions of each node. If a node misses two transmissions, then the cluster head considers it as dead, and communicates this fact to the rest of the zone. The cluster head itself only remains in its position until its energy level drops beneath a certain user defined threshold. At this point, the node with the highest energy level in the network will then elect itself as the new cluster head.

This scheme manages only node failures. It does not provide any functionality, and depends on difficult to measure values, such as energy levels, to make decisions. The same goals of

reconfiguring the network in the event of node failures are handled by routing protocols such as VIBE [PNMP12].

2.7 The WSN Management Problem

In order for a WSN operating system to manage a WSN, information about each and every sensor node needs to be communicated around the network, but in WSN communication is the largest consumer of energy, and therefore needs to be kept to a minimum. We refer to this dilemma as the WSN management problem. This thesis provides a solution to that problem.

We have looked at various approaches to the WSN management problem and these approaches fail for two distinct reasons. The first is that they use multiple, unrelated protocols to provide their services. The Nucleus WSN management system uses both SNMS for query dissemination and Deluge to disseminate new software code images. SNMS builds a routing tree every time a query is made to find the correct node or set of nodes to whom the query is relevant. This protocol is similar to CTP in its requirements. Deluge uses epidemic propagation. There is the possibility that the communication requirement of one protocol will cause starvation and failure of the other protocol. If any of the protocols require a high communication rate then the possibility of protocol failure will increase.

The second reason that some WSN management approaches will fail is that they assume the existence of a central, higher capacity computer node. This assumption will not scale in network population. As the number of sensor nodes increases, then the central node will be unable to receive data from, or communicate to all of the nodes. TASK/TinyDB, TinyCubus, MiLAN, and ZFTMA all exemplify the problem with WSN management approaches that use a centralised abstraction. In all cases they use complex logic and processing to organise the WSN nodes. In the case of TASK/TinyDB there is one node to send and receive queries. MiLAN requires a central node to hold a graph of all of the possible network states and make optimal decisions about the performance of the many protocols it uses to provide services. Both WSN management systems are at risk of failure if the central nodes fail, and will be unable to support

large WSN network deployments.

It is clear that what is needed for a WSN operating system is a way to enable the nodes to provide multiple services in an energy efficient way while being robust to any single source of failure. The core services required include synchronisation so the WSN nodes can act at the same time, and information dissemination, so that the nodes can ensure that they are in a uniform state. Key to this is the ability of the nodes to intercommunicate. However, intercommunication is expensive for WSN nodes, so it needs to be kept to a minimum.

2.8 Requirements for a Solution to the WSN Management Problem

Based on the discussions presented in the previous two sections, I present a series of requirements that need to be met in order to solve the WSN management problem.

1. The services do not interfere with each other.

2. The services have a low communication overhead.

3. The services are provided in a completely decentralised way, and can be controlled from any point.

4. The services are robust to node failures and network dynamics.

These are the basic set of requirements which a WSN operating system needs to provide in order to solve the WSN management problem.

2.9 Our Approach to a Solution to the WSN Management Problem

Based on the requirements given above, I believe that the best approach to solve the WSN management problem is to have a network of sensor nodes who can manage themselves without any central control while using as little communication as possible. We advocate the use of bio-inspired algorithms to enable the nodes to self organize in a completely decentralised manner. This is not enough, however, because bio-inspired protocols are communication intensive, and they need to be used in a way that does not require excessive communication or interfere with other protocols. With the requirements of low communication and no inter-protocol interference, a WSN operating system needs to provide a global set of services so that the sensor nodes may behave like a coherent system and be usable to one or more applications. What is clearly lacking in all previous work except for MiLAN is that the communication medium is one of the most important things that needs to be managed so that higher level services like synchronisation and dissemination can be provided by the system.

2.10 WSN Management Conclusion

Throughout my look at WSN, from the operating system level on the node to the management system level covering the entire WSN, there is a layer, between the application and the network as a whole, that is missing. There needs to be a distributed operating system which manages the information needed by the sensor nodes and abstracts the functionality of the entire WSN. This would provide the WSN's services to an application. TinyDB is a notable exception. Its problem is that it provides a database abstraction, limiting the application to one that uses a data base. This abstraction is limiting because it is centralised, and comes with the limitations already discussed. TinyDB also puts all of the processing on to the base-station. This limits possible applications on which TinyDB can be used, such as those that might have the nodes respond to the data they are sampling, or if the sensors are attached to actuators.

Wireless sensor management and the organisation of wireless sensing nodes into a coherent system is a difficult problem. Sensor nodes work together as a system through communication. But, the communication needed is very expensive for power constrained nodes. We refer to this problem as the WSN management problem.

Bio-inspired algorithms are the best way to enable the sensor nodes to self organise. We view the very act of self organising as a form of biological computation, whose results can be used as system level services. We discuss bio-inspired algorithms and biological computation in the next chapter. The use of bio-inspired algorithms still means that I have to address the WSN management problem.

To deal with the WSN management problem, I consider it important that there be only one single concept which is used to form a system. Multiple organisational concepts may require too much communication and interfere with one another. This point is illustrated later in the thesis, when I try and get communication protocols with different theoretical bases to co-exist on a single node. After this I show how I use just one concept, that of epidemic communication to provide several different network services which co-exist on the same node, and do so with a very small communication cost and address my WSN management problem.

Chapter 3

Biologically Inspired Algorithms and Biological Computation

3.1 Introduction

The WSN management problem is about the movement and processing of management information to every sensor node in the system. Bio-inspired computing gives us an approach to solve this problem. In nature there are systems made up of large numbers of loosely coupled, low resource individuals like flocks of birds, or swarms of fireflies. These systems show organised behaviour: the flocking of birds to escape a predator; or the synchronous flashing of Malaysian fireflies. These phenomenon have been shown to occur without any central leader or control. The process by which these and other systems self-organise is referred to as biological computation. We aim to mimic this ability to self-organise without central control as a solution to the WSN management problem. This is possible because WSN systems can, like some biological systems, treat management information in the same way.

In computer science, there exists the field of bio-inspired algorithms which uses models of the above mentioned natural phenomenon to solve problems. Bio-inspired algorithms use information in a way more suitable to solve management problems in WSN because: they represent information in a distributed fashion which is similar to the way information is represented in

WSN; they read and write information via unreliable communication links which are similar to the unreliable low-powered radios used by wireless sensor nodes; Bio-inspired algorithms process information in a distributed way which is both similar to the way WSN can process information, and has some desirable traits like robustness and scalability which I would like to see in WSN management systems; the notion of information meaning is similar in Biological Computation as that in WSN; and lastly, Biological computations and WSN management have the same notion of termination (or a lack of it). This section gives examples of biological computation, shows the similarities with WSN, and provides examples of bio-inspired algorithms to illustrate these ideas. At the end I discuss the problems associated with adapting protocols used by living creatures for use on wireless sensor nodes.

3.2 Bio-inspired Computation

Bio-inspired algorithms are a branch of A.I. which use mathematical models based on the behaviour of biological systems [OZ06]. Phenomenon like the synchronised flashing of fireflies in South East Asia, the path finding abilities of ant colonies, flocking birds, and the spread of viruses have all been modelled mathematically. These models provide algorithms which can find heuristic solutions to difficult problems like the travelling salesman, and database consistency.

What is captured in these models is the bottom-up, decentralised form of self-organisation observed in organisms such as flocking birds or ant colonies. These systems function without a central decision maker. When birds flock there is no central bird telling each bird where it will be positioned in the flock, or determining the distance between flocking birds. The queen ant in an ant colony only produces ant larvae. There is no central point of command for the ants. The organisation of an ant colony and of a flock of birds comes from each individual ant or bird following the same basic set of rules with respect to its own position or role, and those of its immediate neighbours. These biological systems are self-organising.

Global self-organisation emerges as a result of biological computation [Mit09]. This form of computation takes the state of each of the nodes as input. Consider a flock of birds. The state

on each bird is the current location, movement direction, and speed. The actual processing of this biological computation occurs on each individual, or bird. The processing changes the value of that state on each of the nodes as output. In the case of birds, each bird changes its future state based on the current states of its local neighbours. Its local position, velocity and direction of movement is averaged together with those of its neighbours. The bird also takes into account its distance to all of its neighbours. Using this information each bird will update its own local parameters of velocity and direction of movement. The final, global state that will emerge as a result of the biological computation is a group of birds flying together, in the same direction, at the same velocity, with the same distance between them. This phenomenon is referred to as flocking.

Biological computation differs from the traditional notion of computation in its relationship to the information it processes. Bio-inspired algorithms function by the movements of information around the network, neighbour by neighbour. As a node receives new local information it processes it and updates its state. Eventually the state of all of the nodes converge to a uniform value. At the global level we perceive an emergent result. The emergent result is a by-product of the algorithms, and not explicitly coded in.

Bio-inspired protocols differ from other distributed algorithms in that they produce computational results at two levels, a node level and at a global level. An example of this is a distributed consensus algorithm. A standard global consensus algorithm can unify all of the values among all of the members of a network. In biological computation, the consensus would be found among all of the nodes, as the solution to a global problem affecting all of the nodes. In the flocking of birds to avoid a predator, the global consensus is that all of the birds in the flock fly in the same direction at the same speed, at the same time. The global result is that a predator has a hard time focusing on any single target, and so has a harder time capturing any individual bird. Bio-inspired protocols specifically function on systems who's members are loosely coupled, have no shared memory, unreliable communication, and are prone to failure. In this work I only explore the ability of bio-inspired protocols to create a consensus and therefore solve the global problem of distributed system management and service provision.

The specific bio-inspired protocols I use to perform biological computation are those related to swarm intelligence. Swarm intelligence models systems of loosely coupled individuals, like fireflies, birds, or the spread of illness or rumours in human populations. These systems self-organise without the use of any central control [BW93]. All of the members are assumed to be the same, and are loosely coupled with unreliable communication links. The results of swarm intelligence algorithms is an emergent pattern, or organisation like the flocking birds example given above. The organisation occurs only through local interactions, and is observable only from a global level, i.e. when all of the members of the system are viewed at the same time. Systems modelled by swarm algorithms are similar to WSN because they assume that all of the entities are loosely coupled, are the same (or posses the same set of states), and process information in the same way.

Biological computation based on bio-inspired swarm intelligence is a form of computation distinct from that used in current computers. Melanie Mitchell explains this difference with a comparison [Mit11]:

> The process of information in traditional computers is centralised(i.e. performed by a CPU), typically serial, deterministic, exact, and terminating (i.e., there is an unambiguous final result of the computation). On the other hand, in biology, information processing is massively parallel, stochastic, inexact, and on-going, with no clean notion of a mapping between "inputs" and "outputs".

She continues by posing four information related questions which point to the differences between Turing based computation, and biological computation:

- How is information represented in the system?
- How is information read and written by the system?
- How is information processed?
- How does this information acquire function, purpose, or meaning?

These questions are easy to answer in the current computing environment. Information can be represented as bits and bytes, or as higher-level data types in programs. Information is read from or written to files using read and write functions with defined, reliable semantics. Processing of the information is performed on one or more tightly coupled CPU's governed by a single scheduler. The process either returns a new value, or it can have a side affect and change the state of the system. The purpose or meaning of information relates to the purpose of the program, but it either returns a result, or in the case of interactive programs, returns to a quiescent state to await more input from the user.

Answering these same questions for a WSN operating system shows the appropriateness of using biological computation through swarm intelligence algorithms to solve the WSN management problem. Information is represented as relevant state values on each node, such as clock values, or sensing frequency. The reading and writing of information concerns the way information is communicated around the network to be processed. Information is processed locally on each node. The meaning of the information exists at two levels, the local node level and the system wide global level. The local node level uses the information of a single node and its local neighbours to become part of the overall system. For instance, the nodes can use synchronisation information to synchronise with all of the other nodes in the network. The system wide global level uses the state information of all of the nodes in the system. A WSN application can use this synchronisation information to create global time-stamps for all of the data from all of the nodes in the WSN. At a global level, the information can be used as a service.

Our goal is to enable the self-management of WSN systems as a result of a biological computation. We show that similarities between swarm based biological systems and WSNs makes the bio-inspired approach the best way to manage WSNs.

Information representation

In a natural system like fireflies, each insect flashes its light with a certain frequency. It also observes the times at which its immediate neighbours flash their lights. This system information

is stored on each insect. The only difference is the value of that information, or the time of flashing.

The representation of system information in a WSN management system is the same as in biological swarm systems. The same state information exists on each and every node. For example, the types of information can be: synchronisation information, which is the value of the local node clocks; and configuration information, such as the frequency of sensor samples, or a piece of metadata describing the code version. With both fireflies and wireless sensors, information is represented as the same type on each individual. The only thing that varies between nodes is the initial value of that information. The initial global values of these states can be considered as the input to a biological computation when the system is first started. Over time the biological computation will change the values on each individual node. The global values of these states will be the output of the biological computation.

We perform biological computations with bio-inspired swarm intelligence algorithms because they represent information in a way similar to WSNs. Other bio-inspired methods, such as neural and genetic algorithms, are not used because they do not represent and use information in a way which is compatible with WSNs. For neural networks information is not evenly spread across the system. The network is viewed as a black-box which changes information which traverses it. The changes are dependant on the route which information takes through the network. The route embodies the logic of the solution. The same information is not held at each point in the network.

Genetic algorithms are used to create a program or solution to a problem by randomly mixing and testing various programs or solutions and evaluating them against a fitness function. The information is different code snippets which are combined in various, random ways and then evaluated to test their fitness. This information set does not lend itself to being distributed, and works better represented in a single body of memory.

To show information representation in WSNs, suppose the nodes in my illustrative smoke monitoring network have three sensors, one for particles, one for carbon monoxide, and one for temperature. We assume that each node may only use one sensor at a time. The id of the

sensor currently in use is the state variable I am interested in managing. From the node's point of view, the information is its own sensor id, and those of its local neighbours. The operating system manages the values of all of the state variables of all of the sensor nodes in the network, or the global state of the system.

The goal of the operating system is to ensure is that if one sensor changes the sensor it is using, then all of the other sensor nodes will do the same. This is to ensure that all of the sensor nodes are sensing the same thing at the same time. Our application is to monitor the movement of smoke particles, or carbon monoxide, or temperature across the entire bar over time. We want all of the samples to be of the same phenomenon, taken at the same time so that I may track its movement over time. This homogeneous state of the sensors is the goal of the bio-inspired algorithm, and the output of the biological computation.

Information Reading and Writing

In natural systems, information is read by each individual from its immediate neighbours. Fireflies watch the flashing of their neighbours. Birds flock by observing the positions, directions and velocities of their closest neighbours. Rumours are spread person to person. All of the communication is local, and none of is assumed to be reliable. A firefly may miss some of its neighbours flashes. As long as it sees enough of its neighbours flashing then it will still synchronise. To receive a rumour, it is unimportant who tells you, as long as you receive it from someone. The connections do not have to be reliable.

The reading and writing of the information in WSNs is performed by broadcast radio communication which defines a set of local one hop neighbours. Radio communication is notoriously difficult to model, and unreliable. Biological systems have already solved the problem of unreliable communication links by using redundant communication as mentioned above. They also mitigate the increased communication cost by using only local information. No effort is wasted in communicating commands from a controller multiple hops away.

The nodes in my smoke monitoring WSN will only need to communicate control information to

their local neighbours in order for them to work in unison with the other nodes in the system.

Information Processing

In natural systems like ant colonies or flashing fireflies, all of the processing of the information happens at the individual instead of at a higher-level single, controller. There is no boss ant allocating jobs to the other ants [GM99]. Ants determine their own tasks based on the jobs being performed by other ants around them. Likewise, fireflies flashing in synchronisation are not following a single master firefly, they each synchronise themselves to their immediate neighbours [Buc88]. It is this node level processing which, when performed by the entire network using a bio-inspired swarm intelligence protocol, allows the nodes to adapt themselves to a coherent state and self-manage.

An important requirement for any algorithm running on a WSN node is that it needs to be simple. This means simple processing, without floating point calculations (there are no floating point units on wireless sensor nodes). Algorithmic simplicity also refers to the amount of memory required for the algorithm to function. Each node only has a small amount of memory, and the processors used by wireless sensor nodes do not provide memory protection. Dynamic memory allocation during run-time runs the risk of writing over program code and crashing the node.

The simple algorithm constraint negates the use of several types of distributed A.I. Planning and predicative methods are by and large excluded because they tend to require large amounts of data over which to make future decisions. Neural networks are also difficult to use because of the space needed to store the network as well as the time needed to train the neural net on each node. Agent based methods, although they have been explored [BS07], are also too heavy weight. This is because of the memory required by each agent, and the communication cost to migrate agents from one node to another.

To best understand the type of information processing possible with biological computation on a WSN it is best to look at a simple model. The clearest model for this type of computation

is cellular automaton [Wol06]. In cellular automaton, each sensor node is represented as a square in a grid (one dimensional cellular automaton are also common, but here I consider two dimensional). Each square in a grid represents a wireless sensor node. Each square can only communicate with its immediate neighbours, the adjacent squares. This can include four other squares, in the case of side contact, or eight other squares if you include edge and corner contact (see figure 3.1). The state of each node is modelled simply as the colour of the node, in the simplest case white or black.

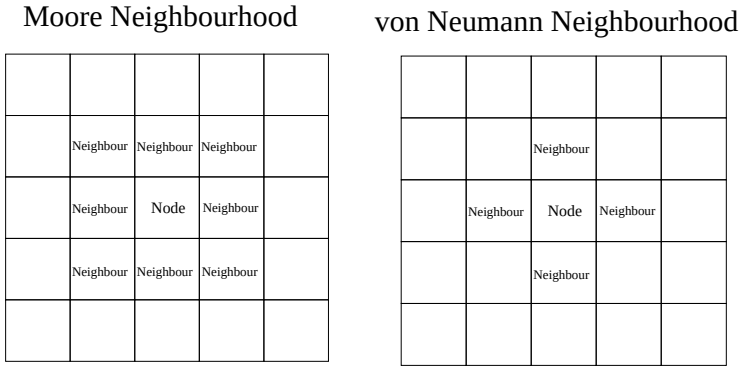

Figure 3.1: Two cellular automata neighbourhoods one with four neighbours, the other with eight neighbours.

In cellular automata models, all of the nodes have the same rules, and update their state based on the states of their neighbours. Different rules lead to different patterns of states over time. Wolfram classifies four different classes of final pattern/results [Wol06].

In class one cellular automata, regardless of the initial state, all nodes converge to a uniform stable state. An example of this is all nodes converging to and remaining at the same colour from an initial random distribution of colours.

Class two cellular automata have a fixed pattern emerging in a fixed region only. This produces a stable, fixed pattern from an initial random state. For instance, a random distribution of colours converge to a fixed pattern like a zebra's stripes.

The third class of cellular automata creates random patterns that do not stabilise in time. This lack of stable behaviour is also referred to as chaotic.

The fourth class is the most interesting. Different initial conditions lead to different patterned

behaviour over time. Unlike class three, there are a finite, fixed variety of patterns which appear, and move across the grid. The regularity of the patterns has lead to their being given names like glider, puffer, or the gosper glider gun. Their location, time of appearance and movement across the grid are unpredictable. It has been shown that the fourth class of cellular automata rules are capable of universal computation [LN90]. One of the best known of these is Conway's game of life [Lan86].

The algorithms I use concentrate only on the use of rules which belong to the first class. These rules allow us to ensure that my smoke monitoring WSN is able to converge to a uniform state, and sample at the same time, with the same sensor, with out needing any central control.

Information Purpose or Meaning

In biological systems I can define two levels of information, with a difference of meaning at each. The low level is the individual level where the information is the local state and the states of local neighbours visible to an individual. A bird would know its own movement parameters, and those of its surrounding neighbours. For a firefly this would be the time it fires, and the firing times of all of its neighbours. The high level information is the states of all of the individuals at the same time, the global view. Fireflies do not stand back from their swarm and marvel at the fact that they are flashing in unison, and birds do not realise the beauty of the flocks they form by not colliding with their neighbours. The purpose, or meaning of information to the users of this information is different at each level.

For WSNs, the node level information is what is processed and adapted to. For instance, in epidemic dissemination of new code, the age of the local code is expressed in a state variable. This state exists on all nodes. It is the desire of the management system that this state variable, and the information which it describes, is the same for all nodes in the network. Every node shares its state variable with all of its local neighbours. The purpose of this exchange is to identify if a neighbour node has newer information. If it does, then the node with older information will seek to update itself with the newer information from the neighbour and update its local state accordingly. The value of a node's local information, that is its state

variable compared to the values of those of its neighbours, indicates whether it is up-to-date or not.

The global level information for a WSN is the value of all of the node's individual state variables at the same point in time. From my epidemic dissemination example, the value of all of the state variables indicates the age of the information for all of the nodes in the network. These values will converge over time to a single, stable value for all of the nodes. This convergence is only observable from a global level. This global state can have purpose or meaning to an application or a user. In the firefly synchronisation algorithm, the global effect of the nodes having uniform state variables is that an observer will see all of the nodes doing something, like flashing their LED's, at the same time.

Biological Computation and Termination

The final difference that I will highlight between traditional Turing-based computation and biological computation is the notion of termination. In Turing-based computation, an algorithm produces a result, and then terminates. For example, the recursive sorting algorithm quicksort will continue until it has no more input to sort, then it terminates returning the sorted input.

In biological systems the equivalent phenomenon is convergence. When a biological computation begins, the states of all of the nodes in the network are the input, and can have any value. Each node runs a simple algorithm, changing its own state based on the state information it receives from its neighbours. Over time, the states of all of the nodes will converge to the same value, and I say that the algorithm has converged. When all of the nodes have converged, this is the equivalent of a termination of a Turing computation. The global state is the output. Using a gossip algorithm as an example, I say that the algorithm has converged when all of the nodes have the newest and the same information.

Convergence also has a relationship with time. There is both the notion of how long it takes to converge, and if the nodes remain converged over time. Time to converge is analogous with time to terminate in traditional computation. This provides a useful and clear metric to evaluate the

performance of an algorithm used for biological computation. The other notion is the stability of the converged state. The algorithm on the nodes continues to run and is ignorant of the global result. So an equally important measure of the success of algorithm used for biological computation is whether the nodes will remain converged over time, and through perturbations, such as node failure. For example, with gossip algorithms I focus on the ability of the network to maintain the newest information. If a new node is introduced to the network, or becomes reconnected, and it has older information, we need to be certain that the old information will not be propagated around the network, and that the new node will eventually have the new information.

One of the great challenges of engineering systems which use biological computation is that it is difficult to mathematically prove that an algorithm will converge, and stay converged. The reason for this is that it is easy to model the behaviour of one node as a dynamical system. But biological computation comprises a multitude of simple nodes, the state of each affecting the state of all of its neighbours. It is this interconnectedness which make analysis of this style of computation so difficult. There currently are no good ways to find closed-form solutions to the global state of highly coupled dynamic systems. The method predominately used to analyse these types of systems is through simulation.

In the work with biological computation which I present in this thesis I measure time to converge through simulation and experimentation. I also determine that the converged state is stable through simulation and via experimentation on WSN test-beds. In one case I use a proof based on a Lyapunov function to suggest stability by showing that queue lengths remain bounded. But, in most cases, my analysis of whether an algorithm will converge and stay in a converged state is done experimentally.

3.3 Bio-Inspired Algorithms Compared with Centralised Algorithms

Bio-inspired computing is a bottom-up, decentralised control architecture as opposed to a top-down, centralised control architecture. Both have their pros and cons, and it is important to describe them here, and doing so will further highlight why I have chosen the bottom-up approach of bio-inspired algorithms to provide WSN management services and deal with the WSN management problem.

The largest difference with these two approaches is that with decentralised control, the processing occurs on the nodes, not at a single location. This gives decentralised control two distinct advantages that are well aligned with the operational requirements of WSNs. The first is that there is no single point of failure for decentralised algorithms, the second is that there are no scalability issues with processing or communication to a central controller.

Environmental WSNs operate out-of-doors and so will suffer from a high rate of failure. In [LBV06] a WSN deployment measuring humidity and temperature in a potato field suffered from many node failures due to leaks in sensor cases, rapid battery depletion due to temperature changes, interference from foliage as the plants grew, and general human interference. The list of reasons for node failures is limited only by the imagination. It takes no great insight to realise that reliance on a single point of control is a very bad policy, regardless of the precautions taken. If the controller fails, then the sensor nodes will remain unmanaged, and the application using the sensor nodes will fail. Bio-inspired swarm intelligence algorithms remove this problem by distributing the control.

To be fair to centralised control, if all of the processing is occurring at a central node, then there is no waiting for a distributed network of nodes to converge. Bio-inspired algorithms take time to converge (their form of termination as discussed above), and not all swarm-intelligence algorithms are guaranteed to converge. For a centralised control system, once the command has been determined, then all that needs to happen is that the information needs to be disseminated to all of the nodes. Bio-inspired algorithms require time for the information in the network to

spread to all of the nodes, and this also affects time to converge.

The other issue differentiating a centralised control approach from the decentralised one which I endorse is scalability. If I process system information on one central controlling node, I will have to collect the relevant system information from each sensor node, and deliver it to the controller. Once the information has been processed, the resulting new system information must be pushed back to all of the sensor nodes. This approach requires a lot of communication. Aside from the communication required is the fact that there will be an upper limit to the number of nodes whose data can be stored and effectively processed at a central controller. If the network grows too large, then the controller may not have the memory to store all of the nodes data, or may be unable to process the new system state fast enough to keep the WSN responsive to change.

Once again, to be fair to centralised protocols, they can be very efficient with their use of communication. If only one node is sending commands, and the commands are disseminated via an efficient routing scheme such as a shortest path tree, then the total communication overhead can be low. This, however, still has a problem with over-tasking the nodes closest to the central controller as routing nodes to the rest of the network. This is a well known problem with WSN systems [PNMP12].

There also exists a class of hybrid protocols that use distributed algorithms to select a central controller. Two examples of these are the Flooding Time Synchronisation Protocol (FTSP) discussed in the previous chapter [MKSL04], and the Low Energy Adaptive Clustering Hierarchy (LEACH) protocol for the formation of WSN clusters [HCB00a]. Both protocols perform distributed leader. In the case of FTSP it is for the selection of a single node to synchronise to. LEACH selects cluster heads among groups of geographically local nodes in order to select a subset of the nodes for routing and control. The hybrid approach has its benefits, that of combining the resiliency to a single point of failure of distributed algorithms with the efficiency of centralised algorithms. The problem with the hybrid approach is that the network can only be in one mode at a time. It will either be acting as a distributed protocol, electing its leader, or it will be functioning as a centralised protocol following its leader. If the leader node or cluster

head fails, the network needs to change modes, and select a new leader. In WSN systems, the failure rate of a node can be high, or the communication links can be unstable. These instabilities can force the WSN nodes to change between centralised mode and leader election mode at a high rate. This may leave the network unable to perform its required task, and can consume a large quantity of communication resources.

Bio-inspired algorithms tend to be robust because of redundancy. The loss of a few nodes or a few messages will not affect the operation of the overall system. The cost of this robustness through redundancy is that there are often a large number of repeated and redundant messages in the system. Redundancy of messages or communication is very undesirable for WSN, as it constitutes a waste of limited communication bandwidth and energy contributing to my WSN management problem. This is the major problem with the use of bio-inspired algorithms in WSNs, and one which I seek to answer in this thesis.

We still feel that the two issues of robustness and scalability are the most important for us in WSN operating system design. This is why I feel it is worthwhile solving the problems of high communication costs and taking a chance on non-deterministic algorithms to use bio-inspired swarm intelligence approach to solve my WSN management problem as a biological computation.

3.4 List of Contributions

One of the challenges in this work is to enable distributed WSN management with a low communication overhead. Central to that is the question of how much, or how little information nodes need to communicate in order to perform the biological computation and obtain the emergent result which they require. What I am concerned with here is not with the quantity of information the nodes can send, as in Shannon's information theory, rather how much, or little, information the nodes need to communicate in order to observe the effect that are required to enable self-management for a WSN.

The reason I need to focus on the amount of information required is that communication in

WSN is expensive from an energy usage point of view. Biological systems do not have the same energy constraints, animals can eat and replenish their energy resources. WSN nodes are battery powered, and communication is their largest consumer of energy. I cannot use the exact same protocols which are observed in nature, because they require too much communication. I aim to try and maintain the robustness and scalability benefits of bio-inspired algorithms while reducing their communication requirement. This will allow me to have self-organising WSN systems with the robustness and scalability of biological systems.

The bio-inspired approach produces algorithms which are simple, robust, and scalable, a perfect fit for distributed systems. There one major drawback is that the algorithms presented to date have not been the most frugal with respect to energy.

The main contribution that I make in this thesis is to provide basic service protocols based on bio-inspired algorithms to form the building blocks of a distributed operating system for WSN. I provide:

1. A bio-inspired cross-layer scheduler which uses only local neighbour information, and local processing to perform a biological computation whose result determines which protocol an individual node should use, and which node should gain access to the communication medium first. A side effect of this computational result is that the nodes are scheduled in a throughput optimal way.

2. A synchronisation protocol which works using epidemic communication which is communication efficient and fast to converge.

3. The ability to provide multiple services through using biological computation using epidemic algorithms in an efficient way.

These protocols give the tools to provide a unified global view of a WSN system through the use of bio-inspired algorithms. The result is a synchronised system with a unified global state which is controllable.

3.5 Conclusion

Information flow and processing are the central elements of distributed system management. Large biological systems like flocking birds or flashing fireflies have similar organisational challenges to WSNs. They are made up of large numbers of low capacity individuals with unreliable connections to one another. I refer to this process as biological computation through the use of bio-inspired swarm intelligence protocols, and argue that it is the best process to use to create self-managing WSNs.

In the next section I look at my first concrete example, and my first contribution. I address the problem of inter-protocol interference. A swarm-intelligence algorithm is presented which uses the protocol queue lengths of the local node and all of its neighbours to schedule which protocol out of several on an individual node may send next, and which node in the local area may access the communication medium. The emergent results of this algorithm is that optimal throughput is achieved for broadcast traffic as long as the protocol requirements are met. Following that, I look at another bio-inspired swarm-intelligence approach to provide the WSN services of synchronisation and information dissemination using biological computation with a low communication overhead.

Chapter 4

Scheduling Multiple Protocols

4.1 Introduction

The smoke monitoring application needs certain services to provide useful data to patrons wishing to avoid second hand smoke, or venue owners wishing to show compliance with health and safety requirements. For this example application, the minimum services needed are: time synchronisation, to make the air quality samples correlate in time; dissemination of commands and parameters around the network, such as which frequency to sample at; and collection to get the data samples to a location where they can be processed. These are the services which which a management layer needs to provide.

Current state of the art WSN deployments use a combination of different protocols to fulfil the above mentioned requirements and provide these services. For example, the TinyOS [LMP+05] operating system comes with a code library, which contains many of the protocols used to enable the aforementioned services. The Flooding Time Synchronisation Protocol (FTSP) [MKSL04] is used to synchronise the sensor nodes and enable them to take time correlated samples. Deluge is a protocol [HC04a] to disseminate updated code images to the entire network, or change parameters on the nodes at a global, system wide level using only local communication between nodes. The Collection Tree Protocol (CTP) [GFJ+09] is a multi-hop data collection protocol to collect sensor data from the nodes and forward it to a base-station for application

processing.

The use of different protocols means that each protocol can be optimised to perform its task in as efficient a way as possible. Although each protocol on its own may be very efficient, when multiple protocols are used, there arises problems with one protocol inhibiting the functioning of the others. Interference can occur at the node level with one protocol starving the other, and at the network level with nodes causing radio communication failure for other nodes. When this happens, a core service needed by the WSN may fail, causing the WSN itself to fail along with it.

4.2 Network Channel Capacity

The combination of common WSN protocols to provide services to applications is problematic. There are examples in the literature of inter-protocol interference occurring in a network, causing poor protocol performance. In the case of an environmental network measuring soil moisture in a potato field [LBV06], periodic heavy communication required by the Deluge reprogramming protocol [HC04a] starved the MintRoute data collection protocol [WTC03] of the beacons it needed to enable data forwarding.

In order to understand this phenomenon of multiple protocols conflicting with one another I set up a WSN comprising 12 sensor nodes and one base-station which measured temperature, and sent temperature readings back to the base-station via a multi-hop network (see figure 4.1). The network used CTP to collect data and Deluge to disseminate code updates (see figure 4.2 for a graphical representation of the application and its supporting software stack). These two protocols were selected because they are readily available WSN protocols, and are representative of the state of the art.

To first create a baseline I measured the percentage of data received by the base-station using CTP on its own (i.e. the only protocol used by the nodes in the network). CTP worked well, averaging 99% data collection at a rate of 25 data messages received by the base-station every minute (the maximum capacity stated in the CTP paper). I also ran the network with just

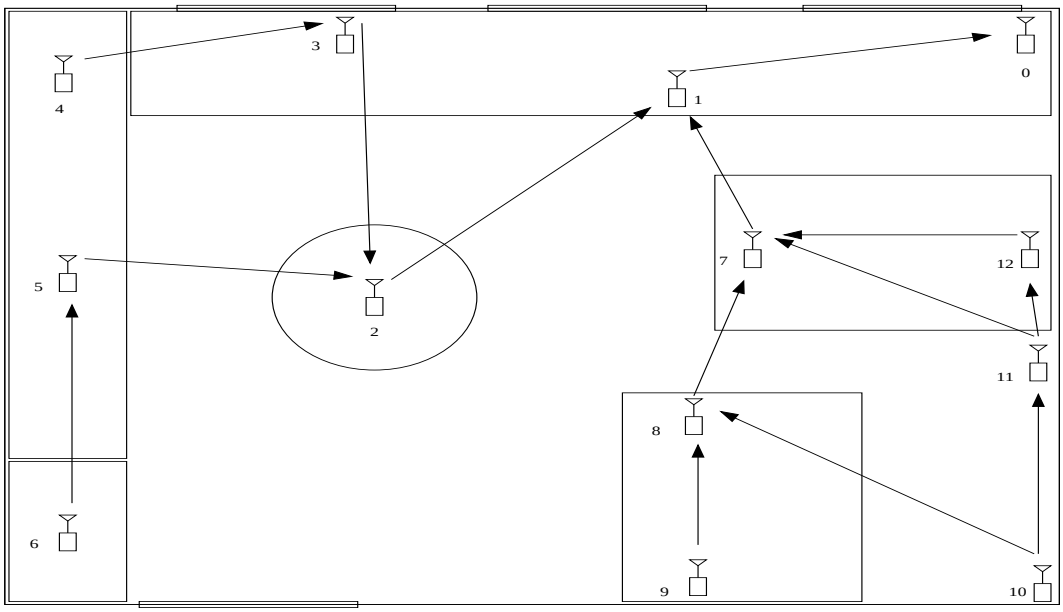

Figure 4.1: Topology of the network used in the experiments. Each sensor node has an id number.

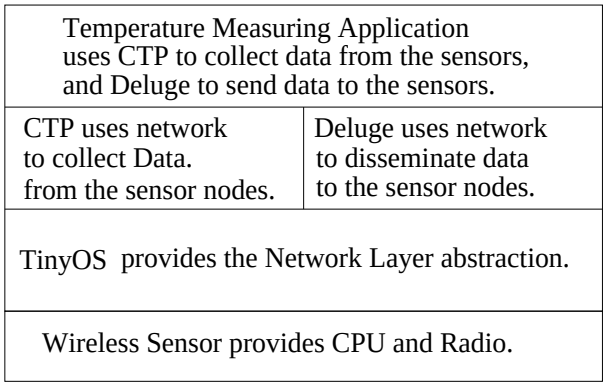

Figure 4.2: A graphical representation of the software stack of the temperature sensing application, the service protocols it depends upon, and the TinyOS operating system.

Deluge (the nodes sampled data, but did not send it to the base-station. I found that Deluge performed very well, and that all of the nodes were updated and rebooted within a maximum time of approximately one minute.

I then combined CTP and Deluge together and looked at the same metrics: data collection rate (for CTP); time to reboot; and percentage of the network rebooted (updated by Deluge). My first attempt was with the nodes sampling and sending data to the base-station once every minute. Deluge updates were sent once every 30 minutes. The data collection rate was still very high, around 99%, but all of the Deluge updates failed to occur during the 30 minute window which they were given to complete. The final update was allowed two hours to complete, and

still failed to occur.

The next trial was to reduce the data sending rate of CTP to once every 10 minutes for each sensor node. Again, CTP worked well at this rate, collecting 99 percent of the data sent by the nodes over a three hour period. This time Deluge also worked, but it took eight minutes and six seconds for its first update and only succeeded in 10 out of 13 nodes used in the trial. The second Deluge use took three minutes and 24 seconds, and 12 out of the 13 nodes rebooted into the new image. All in all, updates were run every 15 minutes, averaged four minutes a dissemination, and 90 percent update success.

It is clear that when two protocols share the same network stack, there is a chance of disruption occurring to at least one of the protocols. This result illustrates the WSN management problem. The problem occurs at one of two areas. The first is the MAC layer, and the MAC protocol is unable to provide the protocols with the bandwidth which they need to function correctly. The other is the protocol scheduler, and the protocols are not being scheduled in an efficient way, causing starvation of one or the other protocol. These two areas are both culpable if the link qualities are good (i.e. each node can hear its neighbours well). The MAC layer issues are a subset of general link quality issues.

In order to test if either layer is more of a problem than the other, another simple WSN was set up. This test counted the number of CCA failures indicating medium congestion, and the number of packets lost in the packet queue due to scheduling.

A smaller network was set up using five sensor nodes and one base-station (see Figure 4.3). The nodes were programmed to indicate via a red LED flash when there was a transmission failure due to medium congestion to indicate MAC problems, and a green LED flash when a protocol was unable to queue a message due to message buffer overflow to indicate scheduler problems. (Note, the AM message layer in TinyOS 2.x only has a buffer depth of one message per protocol). The number of LED flashes which occurred were recorded. The network ran the same temperature measuring application as in the previous experiment and used CTP to forward the data and Deluge to provide new code updates to the network.

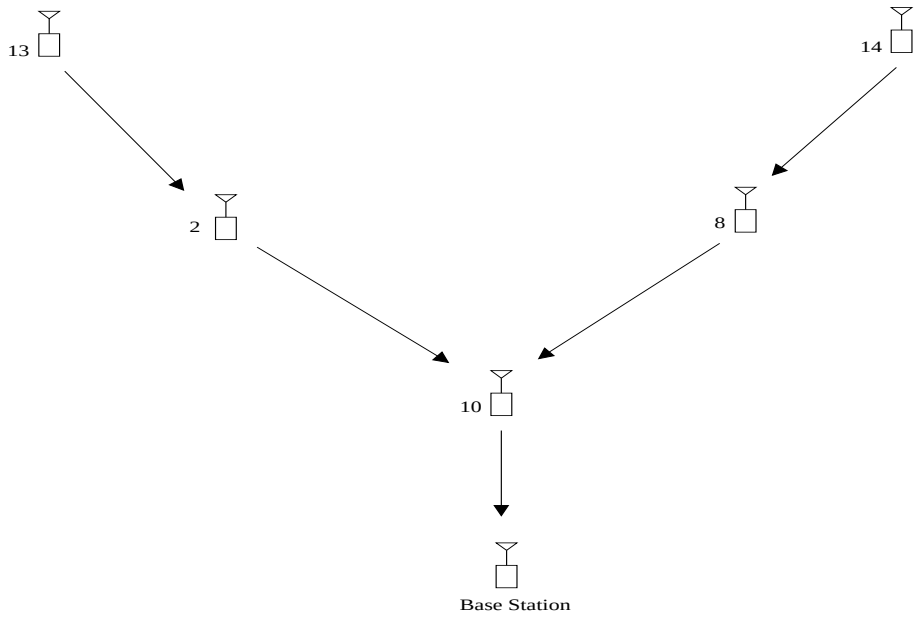

Figure 4.3: Topology of the network used in the experiments. Each sensor node has an id number.

The results showed us that at the upstream routing node number 10 and the base-station there were no MAC layer problems or scheduler problems during steady state CTP operation, that is when there were no Deluge updates sharing the communication medium. As soon as Deluge was used to disseminate and reboot the network, both MAC layer and scheduler problems were observed. The occurrence of MAC layer events was greater than scheduler events by a factor of four. In the worst case the MAC layer indicated 11 failures while the scheduler had four lost packets, and this was observed at node 10. More common results were four to six MAC layer errors with one or two scheduler errors. Errors occurred in 100% of the trials run with a data rate of a packet every 20 seconds or more. Trials performed with a data rate of less than 10 seconds failed to provide the bandwidth needed for both protocols, and Deluge failed.

The RPL IPv6 WSN network protocol [TED10, KTDH$^+$11], discussed in the background chapter, is currently comprised of a routing protocol using the same path cost metric (ETX) as CTP, and disseminates network information using a mechanism very similar to Deluge (using Trickle). As can be seen by these experiments, under a heavy data routing communication load the bursty traffic requirements of Deluge can be starved by the routing protocol. These observations make the authors question the ability of RPL to be used in a WSN application with several different service protocols sharing the network, and potentially high traffic loads.

Variable	meaning	expressed as
G	fully connected, directed graph	$G(N, L)$
N	set of all sensor nodes	$x \in N$
L	set of all directed links between sensor nodes	$(x, y) \in L$
t	time	$t = 1,\ 2,\ 3...$
x	sensor node	$x \in N$
y	sensor node	$y \in N$
(x, y)	directed link from x to y	$(x, y) \in L$
p	a protocol	$p \in P$
P	set of all protocols	$p \in P$
Q	queue length	$Q_x^p(t)$
$\boldsymbol{Q}(t)$	all queue lengths on all nodes	vector
r	rate at which a protocol produces new messages	$r_x^p(t)$
$\boldsymbol{r}(t)$	messaging rate of all protocols on all nodes	
c	link capacity or data forwarding rate	$c_{x,y}(t)$
CB	Broadcast capacity of a node during a time slot	$CB_x(t)$
f	data of a specific protocol broadcast by a specific node	$f_x^p(t)$
S	contention-free transmission vector	S^x
Π	link layer rate region	S^x
a	long term frequency of transmission for each node	a^x
$\bar{\boldsymbol{f}}$	vector of total long-term data rate required by each individual node	
$\mathbb{E}$	the expected value	

Table 4.1: Nomenclature used in the Network Capacity definition

4.3 A Formal Definition of the Problem

The problems observed above are clearly the result of network congestion. This occurs when the communication medium (radio in the case of WSN) has no more capacity to give to the nodes. In order to understand the notion of network capacity a one-hop network is defined as a fully-connected, directed graph $G(N,\ L)$, where N is the set of all sensor nodes and L is the set of all wireless links. Each individual node is described as x in the case of one arbitrary node, and (x, y) denotes a directional link from node x to node y. The nodes x and y are both in the network, $x \in N$ and $y \in N$. To a link from node x to node y is denoted as $(x, y) \in L$. Time is divided into equal length, non-overlapping time periods $t = 1,\ 2,\ 3, ...$

Since there are multiple protocols, CTP and Deluge/Drip in the experiments above, I assume a set of protocols P. All of the nodes in the network use all of the protocols in P, each individual

protocol is referred to as p. Each node maintains a queue Q for each protocol in P. The queue backlog of protocol p at a node x during the time period t is $Q_x^p(t)$, and is expressed as an integer number denoting the number of messages in the queue.

In order to represent all of the queue backlogs for all of the protocols on all of the nodes during a give time period a vector of dimension $|N| \times |P|$ is used which is expressed as $\boldsymbol{Q}(t)$. At time period t, every protocol $p \in P$ used on every sensor node $x \in N$ inserts new messages into its queue at a rate expressed as $r_x^p(t)$. The rate $r_x^p(t)$ is identical and independently distributed (i.i.d.) over each time period t, with a finite second moment $\mathbb{E}[(r_x^p(t))^2] \leq (\mathrm{r}^{\max})^2$. The rate of message production of all of the protocols on all of the nodes at a given time period is the vector $\boldsymbol{r}(t)$ which is of dimension $|N| \times |P|$.

The capacity of a wireless link $(x, y) \in L$ at time period t is denoted as $c_{x,y}(t)$. I assume that $c_{x,y}(t)$ is identical and independently distributed (i.i.d.) over t, with a finite second moment $\mathbb{E}[(c_{x,y}(t))^2] \leq (\mathrm{r}^{\max})^2$.

The local broadcast capacity of a node $x \in N$ is the rate $c_{x,y}(t)$ of its worst link (worst in terms of lowest rate) out of all of the links it has with all of its neighbours.

$$CB_x(t) = \min_{y \in N - \{x\}} c_{x,y}(t), \forall x, y$$

From time period t to $t+1$, the queue length increases by the equation

$$Q_x^p(t+1) = \max(0,\ r_x^p(t) - f_x^p(t) + Q_x^p(t))$$

The function $f_x^p(t)$ expresses the amount of data of protocol p broadcast by the node x during the time period t. This amount of broadcast data ($f_x^p(t)$) will be less than or at most equal to the local broadcast capacity ($CB_x(t)$) of node x.

It is easy to verify that

$$\sum_{p \in P} f_x^p(t) \leq CB_x(t)$$

To avoid message loss due to simultaneous transmission of multiple nodes, only one node in N can transmit during a time period. This is because the local area network $G(N, L)$ is fully connected. Any other nodes transmitting during that time period will cause a collision.

Every node x has a contention-free transmission vector of N dimensions called S^x. Each value of the vector is the CB_x of a given node x for all nodes in the network, including the local node. The x^{th} entry of the vector is for node x and is the broadcast capacity for that node. For each node, only its own location in its contention-free vector has a value (CB_x), all of the other entries for the other nodes are zero. This signifies the contention-free nature of this vector. Only one node, x, can broadcast. In order to avoid contention all of the other nodes must remain silent and have CB_x equal to zero.

The link layer rate region Π is the convex hull (or envelope) over the N dimensional space of all of the contention-free transmission vectors (S^x). The space is the region where all nodes can broadcast without causing contention with one-another. The term a^x is the long term frequency at which a node is broadcasting.

$$\Pi = \{S \mid S = \sum_x a^x S^x,\ a^x \geq 0,\ \sum_x a^x = 1, a^x \in \mathbb{R}\}$$

No combination of node sending schedules which is outside of the space contained within the convex hull Π can be scheduled by any scheduling policy. There will be contention and message loss.

The network capacity region is the set of all input rates that the network can stably support considering all possible scheduling and routing algorithms.

The network capacity region is the set of all of the rates at which all of the protocols add new messages into their queues ($\boldsymbol{r}(t)$) which can be scheduled without allowing the queue sizes to increase to infinity. The queue sizes will increase in an unbounded way, or messages will be lost in a real implementation, due to the fact that the network can not handle all of the communication required by the protocols. This is exactly the problem witnessed in the

experiments above. When the Deluge protocol began to send data it found that the network capacity was not available to it, so it failed.

The average data rate required by every node in the network is a vector $\overline{\boldsymbol{f}}$ of size N. The values of vector $\overline{\boldsymbol{f}}$ are the long term data rates required by that node. The value of the x^th space is for node x. The long term data rate is the sum of the long term data rates of all of its protocols p.

$$\overline{f_x} = \lim_{t\to\infty} \frac{1}{T} \sum_{p\in P} \sum_{t=1}^{T} f_x^p(t)$$

The network capacity region is formally defined as Λ and say that the data rates required by all of the protocols are in the capacity region $\boldsymbol{r} \in \Lambda$ if there is a scheduling algorithm which can provide the required capacity.

$$\overline{\boldsymbol{f}} \in \boldsymbol{\Pi} \tag{4.1}$$

$$\mathbb{E}[f_x^p(t)] \geq \mathbb{E}[r_x^d(t)] \quad \forall x \in N,\ p \in P, \tag{4.2}$$

These two conditions mean that as long as the requirements of all of the protocols are in the capacity region, then it is possible to schedule them, and ensure that the protocol's queues do not overflow and no messages are lost. If the communication requirements of any of the protocols pushes the total data rate $\overline{\boldsymbol{f}}$ outside of the convex hull of contention-free schedules, then no scheduling policy will be able to prevent message loss. If the protocol is not robust to long term message loss and communication failure, like observed with Deluge when CTP was sending at a high data rate, then the protocol will fail.

4.4 Approach to WSN Service Provision

My first attempt to make a WSN operating system is to try and use the pre-existing service protocols available in the TinyOS library. The aim is to develop a scheduling policy which will ensure that each protocol used in a network can function correctly, subject to the capacity region constraint. In order to solve this problem I need to address two scheduling problems. The first is to schedule a protocol's radio usage on an individual node so that it continues to function. The second is to schedule a node's access to the wireless medium so that it and its neighbours use as much of the medium as possible while reducing the idle back-off time used by CSMA to avoid communication collisions.

For reasons given in chapter 3, I want to solve these scheduling problems in a bio-inspired way. I realise this by using a completely decentralised approach, and make decisions using only local information. The aim is to use biological computation, where the emergent result is a distributed schedule that will enable multiple protocols to function. The result is that the communication channel is used as efficiently as possible, and allows each protocol to function unhindered by the operation of another protocol. This chapter is based on work published with Shusen Yang in [BYM13]

4.5 State of the Art

The cause of the problem of inter-protocol interference is that the current state of the art in WSN systems concentrates on the operating system at the node level, or the MAC layer, ignoring the possible effects of multiple protocols on multiple nodes and their behaviour at the network/system layer. As communication is the key resource which needs to be managed, I propose an enhanced protocol and medium scheduler which uses only local network information (one hop neighbours) to make globally throughput optimal decisions for the whole network.

Current operating systems such as TinyOS provide the abstraction of a private radio to each protocol. Each protocol has a one deep message buffer which is scheduled on the radio in a

round-robin fashion. If a protocol has a high data rate it may fill its one message buffer faster than the radio can send the message. When this occurs, the protocol loses its message and it receives an error from the radio stack indicating that the radio is busy.

The Fair Waiting Scheduler (FWS) has been proposed [ICKJL09] which ensures that each protocol gets the same amount of radio access measured in usage time (both messages sent and received). This protocol also modifies the CSMA layer by adding a 'grant to send' semantic (GTS) which has the effect of increasing sending delays to reduce collisions and ensure that protocols function isolated in time.

FWS uses two mechanisms to reduce inter-protocol interference: GTS and fair queueing. GTS uses protocol isolation to ensure that only one node in a local area uses the communication medium at a time. Protocol isolation works by including a special time-duration field, referred to as the grant duration, in every link-level packet. The grant duration is the period of time during which the receiver of the packet has exclusive use of the communication medium. It operates as a lock and allows the receiver time to either respond to the transmitter, or forward the packet to another node without the risk of loosing the packet to communication collision.

The other mechanism employed by the FWS is fair queueing. The communication usage of each protocol on a queues is monitored. Each time a protocol sends or receives its communication usage in time is recorded. This time includes the airtime of the communication, and the length of the grant from GTS. When there are multiple protocols wanting to send a packet, the one with the lowest usage is allowed to send first. The usage table is allowed to decay with time, so that a protocol with a bursty communication pattern is not penalized during periods of low communication.

The problem with this approach is that it does not take throughput into account. The result is that protocols are isolated and function well at low communication rates, but as the communication rate increases, the fair scheduling may be unable to provide sufficient bandwidth for one of the potentially many protocols to function properly. The data rate that a local area network (one-hop) can handle is affected by both the data rate of individual nodes and the density of nodes. If either increases to a certain point, then the capacity of the communication medium

(radio) will saturate, and data will be lost.

Another approach is to try and reduce the broadcast overhead of service protocols. Most WSN protocols use some broadcast messages to maintain and manage the services they provide. Time synchronisation protocols like FTSP use broadcast packets to synchronise the network. Dissemination protocols such as Deluge use broadcast packets to inform neighbour nodes of new information. Collection protocols including CTP use broadcast to inform neighbour nodes of routing possibilities. Two solutions attempt to reduce broadcast message usage to reduce communication bandwidth consumption.

The Unified Broadcast (UB) adaptation to the TinyOS stack [HJK11] allows protocols using broadcasts to combine them. The Unified Broadcast layer is a layer between the application layer and the network layer where all broadcast messages can be combined into one large message. This layer handles the marshalling and unmarshalling of the data and the delivery of the data to the correct protocol. It is transparent to the application layer protocols. This approach successfully reduces the number of overall transmissions in the network, and frees up bandwidth accordingly. The total number of transmissions in the network is reduced, but does not handle the problem of interfering protocols.

In a system using UB, broadcast protocols submit their packet to the network layer for transmission. The network layer identifies the packet as a broadcast packet, and store it in a buffer. When the UB buffer is full, it sends the large packet consisting of several broadcast packets to the radio layer to be broadcast. When a node receives a UB packet, the radio layer passes it first to the UB layer, which then unpacks the message, and notifies each of the corresponding protocols that a packet has arrived. In the implementation evaluated in [HJK11], the authors increased the default packet payload size of the TinyOS packet from 28 bytes to 60 bytes in order to enable the protocol to aggregate more packets. A special mechanism was also introduced to allow broadcast protocols which had latency requirements to be sent immediately, by forcing the send of the UB buffer.

A similar solution to UB is called the announcement layer [DMT+11]. It provides a service where protocols can request that their broadcast packets are sent at a certain rate, and then it

combines all of the broadcasts into as few transmissions as possible. This approach also only handles broadcast transmissions, and does not deal with the more general problem of interfering protocols.

4.6 Proposed Solutions

The solution that proposed is a radio scheduler which combines medium access control and protocol scheduling. The scheduling policy uses a node's link quality with all of its neighbours and the queue length of the different protocols contending for radio use to determine the next node and protocol to send. This protocol is described in figure 4.5. Expected transmissions is used to determine link quality to deliver a packet to a one-hop neighbour (ETX) [FGJL07]. In practice any gradient based route cost metric will work.

4.6.1 The Scheduling Algorithm

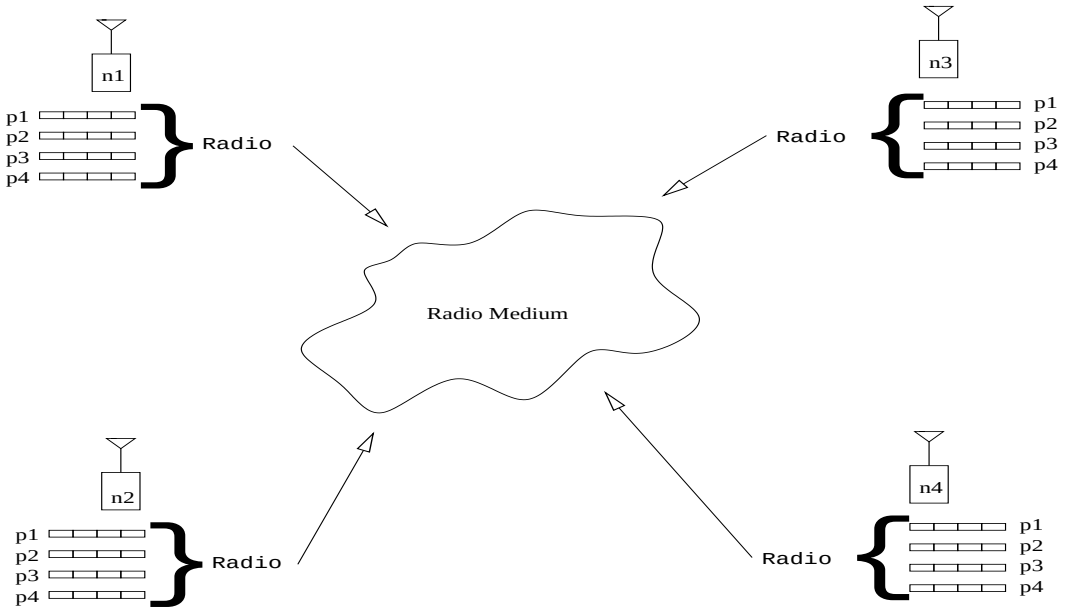

Figure 4.4: Visualisation of the network with multiple protocols per node.

Recall the network model from the beginning of this chapter. This network is visualised as the one-hop network depicted in Figure 4.4.

$Q_i^{longest}(t) = \max p \in P\ Q_i^p(t)$
$C_i^{min}(t) = \min j \in n_i\ C_{(i,j)}(t)$
$w_i(t) = q_i^{longest}(t) * C_{(i,j)}^{min}(t)$
$BROADCASTw_i$.
if $w_i \geq w_j(t), \forall j \in n_i$ **then**
  $CSMA_{backoff} = CSMA_{backoff}/10$.
else
  Do nothing.
end if

Figure 4.5: Greedy Queues: a combined MAC and scheduling protocol

Assume that time is divided up into time periods $t = 1,\ 2,\ 3, ...$, and at the beginning of each time period each node in the network $x \in N$ will compute a weight made up of its broadcast capacity $CB_x(t)$ times the longest queue length of all of its protocols $\max_{p \in P} Q_x^p(t)$.

$$w_x(t) = CB_x(t) \max_{p \in P} Q_x^p(t) \tag{4.3}$$

In any given time period t , assume that every node x will have a protocol with more messages in its queue than any other protocol on that node. If two or more protocols have the longest queues, then chose one at random with a uniform probability. If no protocol has any messages to send, then that node's weight is zero.

The protocol with the most messages in its queue (indicated by the star superscript) is referred to as $p_x^* = \arg\max_{p \in P} Q_x^p(t)$ where $p_x^* \in P$.

At the beginning of a given time period t, every node x will multiply its longest queue length q_x^* by its $CB_x(t)$ to get its weight $w_x(t)$. This weight will then be communicated to all of the other nodes in its one hop neighbourhood. At the beginning of the next time period $(t + 1)$, the node with the highest weight $x^* = \arg\max_{x \in N} w_x(t)$ will broadcast the next message from the protocol which had the longest queue length p_x^* immediately. The rest of the nodes in the network will broadcast the next messages from their longest queues q_x^* after the standard random CSMA backoff period. The node with the highest weight broadcasts before the shortest backoff time receivable from the random CSMA backoff time. This means that the amount of

data broadcast by protocol p_x^* will be

$$f_x^p(t) = \begin{cases} \min(CB_x(t), Q_x^p(t)) & \text{if } x = x^*,\ p = p_{x^*}^* \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

4.6.2 Performance Analysis

This scheduling policy will converge to the point where the populations of all of the queues in the network do not exceed their limits, and that no protocol has to drop packets. This policy can maintain this stability only as long as the data sending rate of the nodes are with-in the channel capacity policy previously defined as $\boldsymbol{r} \in \Lambda$.

Theorem 1. *Given arrival traffic $\boldsymbol{r}$ such that $\boldsymbol{r} + \epsilon \in \Lambda$ for some $\epsilon > 0$, the scheduling scheme will stabilise the network.*

Proof. To prove that the protocol queues will remain bounded it will be shown that the maximum value of the queue lengths is bounded. This is done by using a Lyapunov function

$$V(t) = \sum_{x \in N} \sum_{p \in P} (Q_x^p(t))^2 \tag{4.5}$$

and consider its conditional expected drift:

$$
\begin{aligned}
&\mathbb{E}[\triangle V(t)|\boldsymbol{Q}(t)] \\
&= \mathbb{E}[V(t+1) - V(t)|\boldsymbol{Q}(t)] \\
&= \mathbb{E}[\sum_{x\in N}\sum_{p\in P}((r_x^p(t) - f_x^p(t) + Q_x^p(t))^2 - (Q_x^p(t))^2)|\boldsymbol{Q}(t)] \\
&\leq |N||P|(\mathrm{r}^{\max} + \mathrm{c}^{\max})^2 \\
&\qquad +2\mathbb{E}[\sum_{x\in N}\sum_{p\in P}(r_x^p(t) - f_x^p(t))Q_x^p(t))|\boldsymbol{Q}(t)] \\
&\leq_a |N||P|(\mathrm{r}^{\max} + \mathrm{c}^{\max})^2 - 2\varepsilon(\sum_{x\in N}\sum_{p\in P}Q_x^p(t)|\boldsymbol{Q}(t)]
\end{aligned}
$$

the inequality $\leq_a$, is because of (4.2) and the max-weight scheduling scheme (4.3). Taking an expectation over $\boldsymbol{Q}(t)$ and a telescopic sum from $t = 1$ to T, yields:

$$
\begin{aligned}
\mathbb{E}[V(T)] - \mathbb{E}[V(1)] \quad \leq \quad & T|N||P|(\mathrm{r}^{\max} + \mathrm{c}^{\max})^2 \\
& -2\epsilon\sum_{t=1}^{T}(\sum_{x\in N}\sum_{p\in P}\mathbb{E}[Q_x^p(t)])
\end{aligned}
$$

Dividing both sides by T and taking a limit superior (lim sup) the long term expected queue length of each protocol on each node is less than or equal to the maximum bound. Remember that the maximum bound is determined by the protocol's message injection rate and the proximity of the maximum injection rate to an injection rate which defines the edge of the network capacity region:

$$
\limsup_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T}\sum_{x\in N}\sum_{p\in P}\mathbb{E}[Q_x^p(t)] \leq \frac{|N||P|(\mathrm{r}^{\max} + \mathrm{c}^{\max})^2}{2\epsilon} \tag{4.6}
$$

As long as all of the protocol's on all of the node's requirements remain in the capacity region, the scheduling policy converges to the stable state where the queue population is a finite number.

The queue population does not increase without bound, and no messages will be lost.

$$\frac{|N||P|(\mathrm{r}^{\max}+\mathrm{c}^{\max})^2}{2\epsilon} < \infty \tag{4.7}$$

This proof clearly shows that by using the Greedy Queues scheduler the long-term average of all message queue lengths will be finite when the message injection rate into the network is in the capacity region.

If the long-term rate at which all of the protocols on all of the nodes inject messages into the network is in the capacity region, the long term expected value (or the long term average) of all of queue lengths for all of the protocols on all of the nodes will be a finite value. What is more, this finite value will be less than or equal to a value determined by the distance from the message injection rate to the edge of the capacity region. This finite value is less than infinity. Therefore this proves that the Greedy Queues scheme is throughput optimal within a given capacity region provided by the system under consideration.

Please note that in the proof a lim sup is used, but it could be replaced by a lim for two reasons. Firstly, I assume environmental monitoring applications with a stable data rate. Secondly, I am only concerned about the upper bound of the long term data rate, not the lower bound, because the lower bound will not cause message queues to overflow.

In the next section I implement the Greedy Queue Scheduler to evaluate its performance on a real WSN network.

4.6.3 Greedy Queue Scheduler Implementation

I implemented the Greedy Queues Scheduler for typical low power sensor nodes using TinyOS [LMP+05] on the MicaZ and Telosb WSN platforms. This was done by changing the default message queue provided by the TinyOS networking stack called the Active Message Layer. In the default implementation, each protocol gets its own queue with a length of one. If the

protocol tries to send a message when its queue is full, it receives an error message indicating that there is no more buffer capacity. The first modification I make to the default TinyOS networking stack is to give each protocol a message queue of 4 messages and implement it with FIFO semantics. The lengthened queue is necessary in order to create the queue-length gradients used by the Greedy Queue scheduling approach. The FIFO semantics was used instead of LIFO semantics to ensure that all messages get sent. In a network with high message traffic all of the queues may get filled up. If this happens, then LIFO semantics would mean that some messages may never get sent (see figure 4.6).

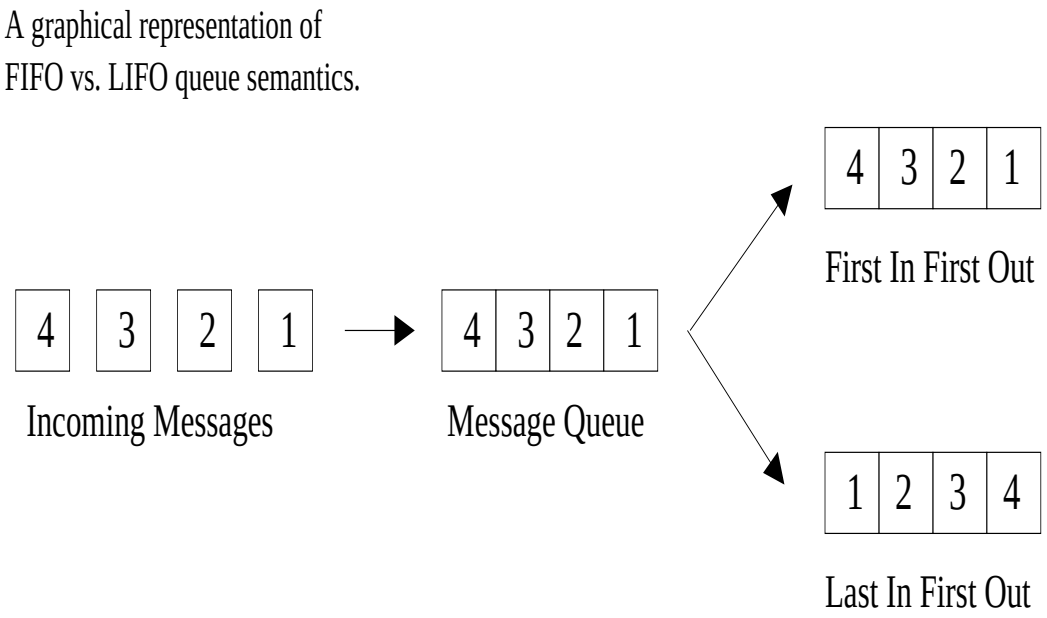

Figure 4.6: A graphical representation of the difference between a First In First Out queue and a Last In First Out queue.

The second modification I make is to change the scheduling policy to the Greedy Queues scheduler mentioned in equation 4.3. This scheduling policy uses an evaluation metric which is referred to as weight, denoted as $w_i(t)$ in figure 4.5. The weight is calculated periodically by multiplying the length of the longest protocol queue with the highest packet reception ratio (PRR, ratio of packets received by a neighbour over the number of packets sent to the same neighbour) I have with any individual local neighbour node. That weight is added to all broadcast message headers and sent to all one-hop neighbours. All nodes in the network use a neighbour table to record the weights received from all one-hop neighbours.

The PRR of all of the links is obtained through the use of the link estimator component available

in the TinyOS libraries. These reception ratios are stored in a neighbour table which is then queried when the scheduler runs. In practice the neighbour table has a maximum capacity of ten neighbours. This makes neighbour table look-ups fast, and reduces the memory consumed by the large data structure.

When a node receives a request to send from a protocol, it runs its scheduling policy. It compares its weight with the weights of all of its neighbours stored in its neighbour table. If it finds that its weight is the largest, it reduces its initial CSMA backoff to a minimal value and sends its message. If it does not have the largest weight, then it sends with a normal randomly chosen CSMA back-off. The short back-off chosen by the node with the highest weight is less than the minimum of the randomly chosen range of the initial CSMA back-off, ensuring that it sends first.

4.7 Evaluation

There are two ways the network medium can become congested, the first is by increasing the population of nodes while maintaining a constant rate of sending for each node. This scenario taxes the collision avoidance mechanism (BMAC using CSMA in this case) used in the network. The other way is to increase the sending rate or the message size of a fixed population of nodes. I examine both scenarios here in order to better understand how radio congestion causes inter-protocol interference, and under what conditions the protocols fail. I also evaluate the solutions to this problem.

I perform experimental evaluation in order to further understand the degree of effect multiple protocols can have on each other, as well as test that the proposed solutions help solve the problem. I use a temperature sensing application made of three service protocols, CTP to collect temperature data, Deluge to disseminate code updates, and FTSP to synchronise the temperature samples (see figure 4.7 for a graphical representation of the application and the service protocols upon which it is built).

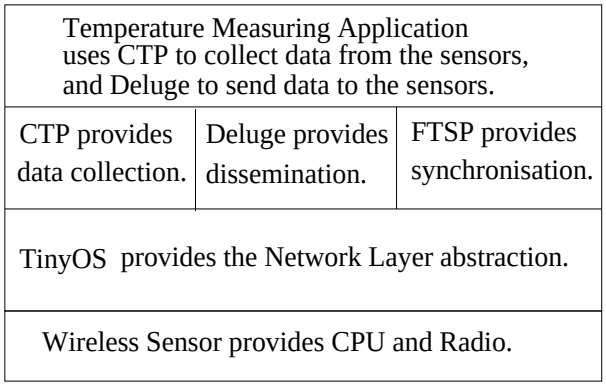

Figure 4.7: A graphical representation of the software stack of the temperature sensing application, the service protocols it depends upon, and the TinyOS operating system.

Parameter	Single-Hop	In-Lab Multi-Hop	Remote Multi-Hop
packet size	128 Bytes (160 Bytes for UB)	128 Bytes	128 Bytes
CTP sending rate	8/sec	8/sec	[1, 1.5, 2, 3, 5, 10]/sec
Node populations	[4, 8, 16, 32]	16	138
Inter-Node distance	2-4 meters	2-4 meters	Indirya WSN testbed
TX power level	0 dBm	-20 dBm	0 dBm
Max Hop Depth	1	4	8

Table 4.2: Parameters for the experiments performed in this chapter.

Performance metrics are evaluated which relate to the protocols used in the network. In the case of CTP, I measure messages received from each node at the CTP base-station. For Deluge I evaluate the time it takes for a new code image to propagate across the network. For the FTSP time synchronisation protocol I measure the variance of the time-stamps received at the base-station.

The first set of experiments was run on a single hop network so that I could increase congestion by increasing the one-hop neighbour population. Analysis was done for a single-hop network, so I start the evaluation there. I used a fixed CTP sending rate as the application traffic, and then added the bursty traffic of the Deluge protocol.

The second set of experiments I perform is on two multi-hop networks, one in a lab, the other in a remote location. The remote testbed I use to increase the application sending rate (via CTP) while maintaining a constant network size.

Together the experiments provide insight into the nature of the problems experienced when multiple protocols are used on the same WSN, and I provide a solution of how to mitigate these problems.

4.7.1 Single-hop results

The first set of experiments were performed on a single hop network. The nodes were distributed around all of the desks and hung from the ceiling of the laboratory, with roughly two to four meters distance between any two nodes (see Figure 4.15 for the arrangement of 16 nodes). The parameters measured were the time taken to disseminate a 38 page (1024 bytes per page) image over a network where CTP is also running at a rate of 8 data messages per second per node. The high data rate was chosen for CTP to saturate the network and evaluate the capacity of our scheduler to continue to provide service to each protocol when there is no excess capacity remaining to schedule. We increased the population of the network by doubling the population of the previous experiment. A node is counted as having received the required Deluge data only if it is able to reboot into the new image. Table 4.2 gives the parameters used in the experiments.

Since these experiments represent the search for a scheduling policy which allows multiple communication protocols to co-exist, it is important to take the three graphs of results together. The first graph (Figure 4.8) shows the average time which the Deluge reprogramming protocol takes to reprogramme the test-bed. The times to disseminate need to be taken with graph (Figure 4.9) which is the percentage of successful Deluge dissemination trials for each scheduler. In cases where no dissemination events were successful, the time to disseminate drops to zero. When no dissemination was successful, Deluge completely failed. The third graph (Figure 4.10) shows CTP success rates as the network becomes congested and Deluge fails.

Deluge works well with all of the schedulers when the local neighbourhood population is only four nodes. The time to disseminate is fairly similar for all of the protocols except the Fair Waiting Protocol (FWP). FWP works, but is almost twice as slow as the others. The percentage of successful disseminations are 100% for all of the scheduling policies, and all of the packets

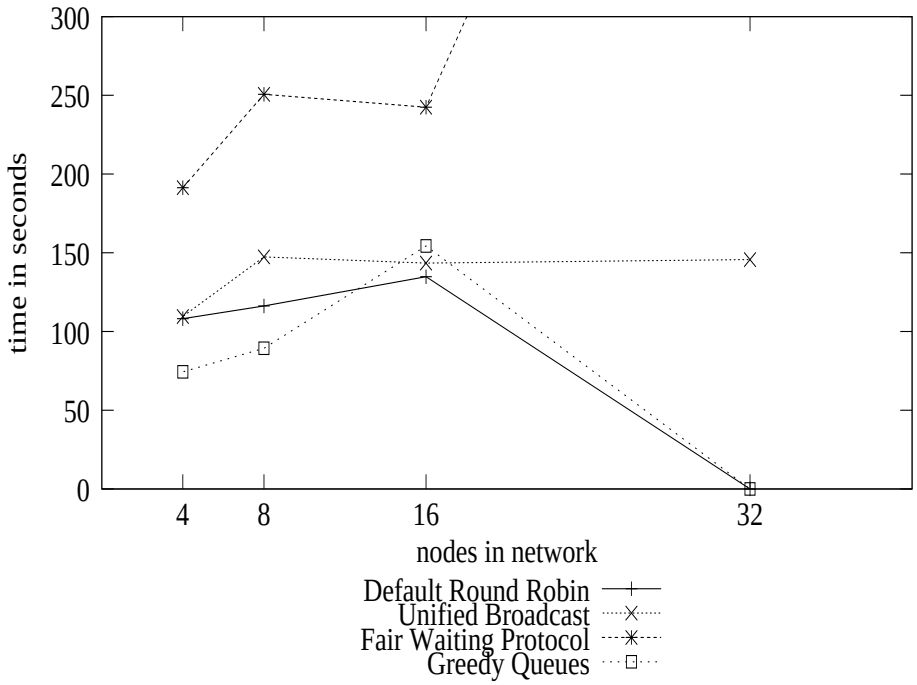

Figure 4.8: Plot of the time for each protocol to disseminate an entire Deluge binary.

sent by the sensing nodes are received by the CTP base station. These results give us a base case, where all of the protocols run well because they are well within the capacity region.

When the population of the network is doubled to eight nodes, the performance of the schedulers begins to vary. The Unified Broadcast scheme hits an upper bound for dissemination time at this point. In subsequent trials its dissemination time remained fixed. The time for FWP to disseminate increases to an average of 250 seconds, taking it off of the graph. The Greedy Queues scheduler has the lowest dissemination time, 10% faster than the default Round Robin scheduler. All of the protocols successfully complete 100% of the Deluge trials except for the default Round Robin scheduler, it only completes 60%. The percentage of data messages received by the CTP base station is still very high for all schedulers, in the 94% range.

At a network population of 16 nodes the performance is still good. The Deluge dissemination time for all of the schedulers remains constant, or increases only by a small amount. The percentage of Deluge trials successfully completed is still 100% for all schedulers except the default Round Robin scheduler, it succeeds in only 5% of its trials. The percentage of CTP data messages received by the base station remains constant from the previous network populations for the Greedy Queue scheduler and the Fair Waiting Protocol, and degrades by about 15% for

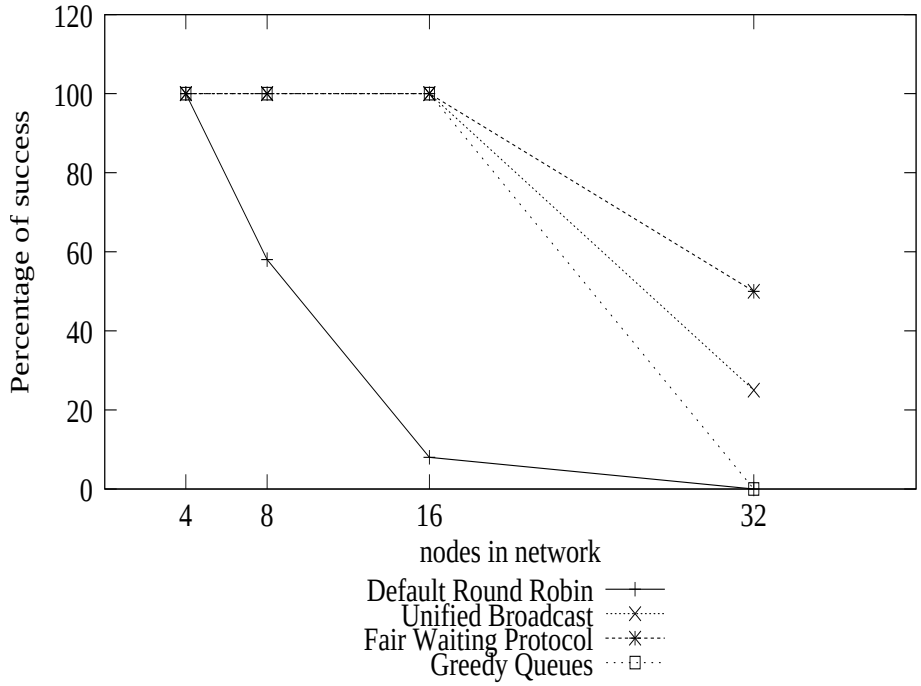

Figure 4.9: Plot of the percentage of trials where the Deluge binary was successfully disseminated.

the default Round Robin scheduler and the Unified Broadcast layer.

At 32 nodes the phenomenon of protocol failure becomes apparent. At this point the scheduler is trying to operate outside of the capacity region, beyond the constraints of any scheduler. FWP's time taken to disseminate Deluge updates increases to almost ten minutes, but only 50% of the disseminations are successful as can be seen in Figure 4.9. Unified Broadcast fares worse with about 30% success, taking the same time as for 16 and 8 nodes. The Greedy Queue and the default Round Robin schedulers have no successful dissemination attempts.

One cause of the Greedy Queue scheduler's failure to disseminate in a network of 32 nodes was the greedy approach to MAC scheduling. Greedy scheduling can cause very high packet loss. LED's were set to blink if there were MAC layer collisions detected, resulting in lost transmissions. The Greedy Queue scheduler showed a large number of lost packets due to collisions when 32 nodes were used. These results were observed during the experiments, but the quantities were not recorded because there were no external devices connected to the nodes to record the failures. Making the nodes record the failures themselves affected the nodes communication performance and skewed the results.

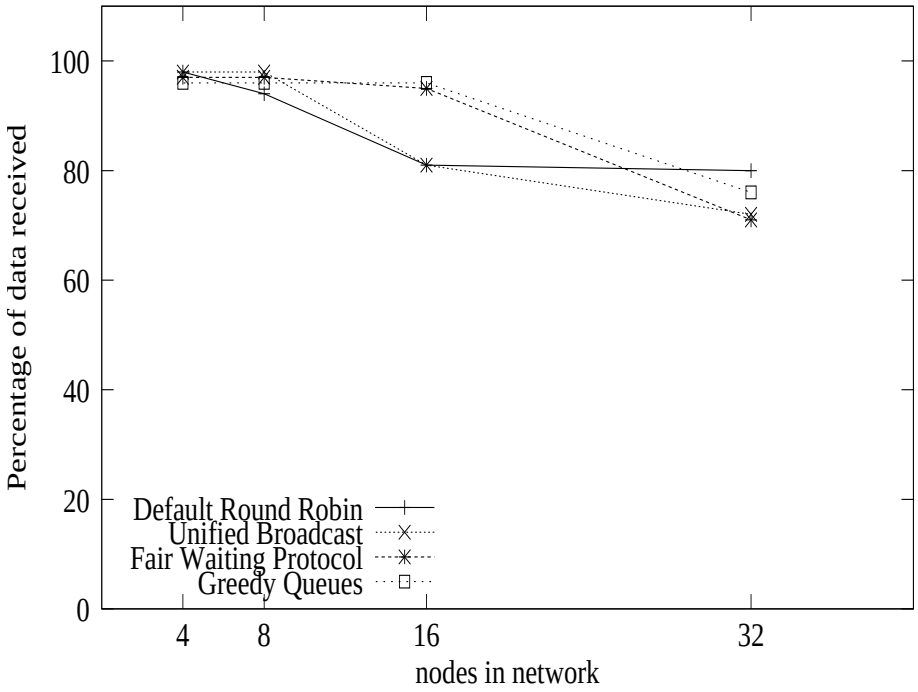

Figure 4.10: Plot of the percentage of data message delivered using CTP during the Deluge dissemination.

It is important to note that at 32 nodes, the network is beyond its capacity region for the purposes of scheduling. This is due to the large number of sensor nodes using the communication medium at a rate beyond which there is capacity. The Greedy Queue scheduler was proved as throughput optimal while within the capacity region. The experimental results show that beyond the capacity region, the Greedy Queues scheduler fails to function well.

In all cases the FTSP protocol functioned properly. This is because in a single-hop environment only the FTSP synchronisation root will broadcast a sync beacon once every three seconds and therefore all of the nodes can synchronise to the same time-stamp.

Looking at the results of the three graphs together, it can be seen that the Greedy Queues scheduler managed 75% data reception, but with a large number of collisions, and the complete failure of Deluge. The Fair Waiting protocol and Unified Broadcast both had 70% data reception averages, and neither could reliably accommodate the communication needs of Deluge.

To address the protocol failure observed with the Greedy Queue Scheduler in a congested network, I decided to make a small modification to the scheduling policy to cope with situations where the network demand of the protocols exceeds the network channel capacity.

$q_i^{longest}(t) = \max mq_i^m(t)$
$C_i^{min}(t) = \min j \in n_i C_{(i,j)}(t)$
$w_i(t) = (q_i^{longest}(t) * C_{(i,j)}^{min}(t)) - q_i$ usage in last n seconds
$BROADCASTw_i$.
if $w_i \geq w_j(t), \forall j \in n_i$ **then**
  $CSMA_{backoff} = CSMA_{backoff}/10$.
else
  Do nothing.
end if

Figure 4.11: Greedy but Fair: a combined MAC and scheduling protocol

4.7.2 The Greedy but Fair Scheduler

It is clear in figures 4.8 and 4.9 that at a population of 32 nodes, the Greedy Queue scheduler fails to provide Deluge with the necessary communication. To explore a simple modification to this situation, I decided to add a fairness penalty to the protocols if they have been using the radio too much. I refer to this policy as the Greedy But Fair scheduler. This use of a fairness penalty is well known in computer science literature and a good introduction is given in [DKS89]. With this scheduling policy I subtract a usage penalty from a protocol's queue length when I calculate the weight. The usage penalty itself reflects usage over time. It needs to be increased with usage as well as decreased (to a minimum of zero) with lack of usage over time. I reduced all penalties every fixed period of time to account for non-usage of the channel.

For the evaluation I maintained a usage counter of each message sent for each protocol to use as a penalty to the weight. This counter was halved every two seconds to prevent the usage counter size from increasing without bound, but still maintain some usage information to inform the scheduling process. This protocol is very similar to the Greedy Queue scheduler, the only difference is that the weight $w_i(t)$ is now reduced by the usage counter. The more messages a protocol sends in a block of time, the lower its weight will be.

4.8 Single-hop Results

I performed the same set of experiments on the same testbed used for the Greedy Queue scheduler evaluation. I plot the results of the Greedy But Fair scheduler against the previous results.

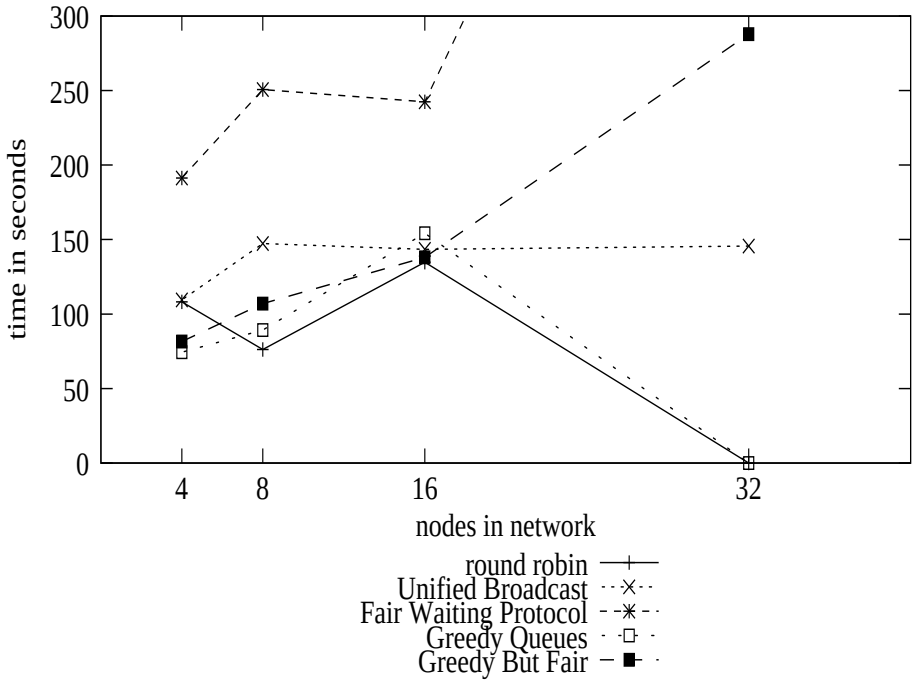

Figure 4.12: Plot of the time for each protocol to disseminate an entire Deluge binary.

The Greedy But Fair scheduler does well in the 8 node network. Its time to disseminate only increases slightly (figure 4.12) from the 4 node network. All of the Deluge trials are successful (figure 4.13). The base-station is still receiving more than 90% of its data (figure 4.14). When the network increases to 16 nodes the Greedy But Fair scheduler continues to cope well. Deluge still functions in 100% of the trials. The dissemination time is slowly increasing, and is now 60% slower than it was in a four node network. The performance of the CTP collection protocol is starting to degrade, but is still above 95%. The Greedy But Fair protocol shows a real improvement when the node population increases to 32 nodes. Its dissemination time increase to 300 seconds, but maintains 100% successful disseminations. This result shows that adding a fairness metric to the scheduler allows it to function better outside of the network capacity region.

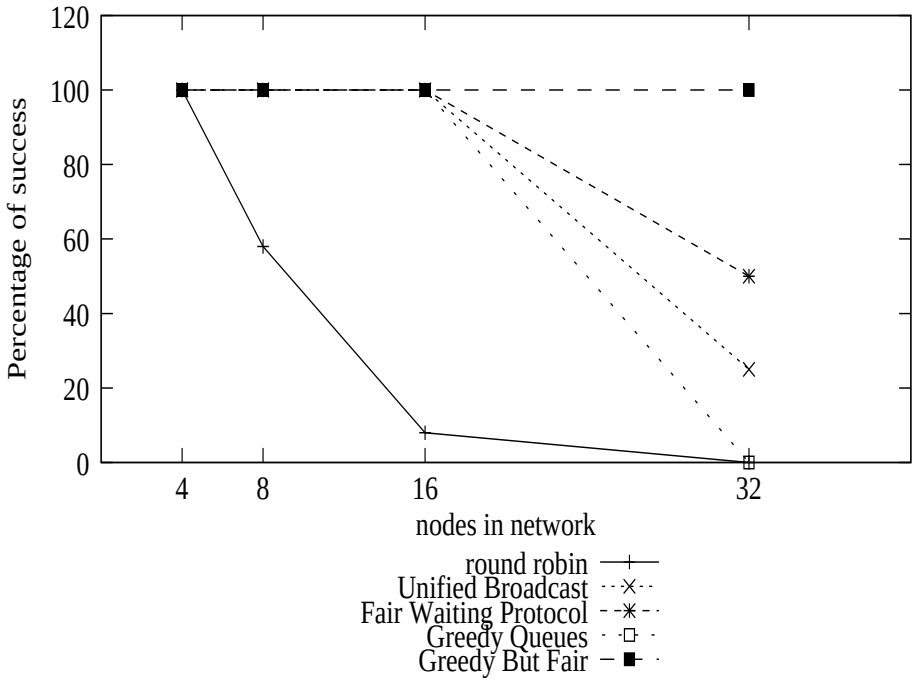

Figure 4.13: Plot of the percentage of trials where the Deluge binary was successfully disseminated.

There are trade-offs with adding the fairness metric. Looking at the results of the three graphs together, it can be seen that although the Greedy But Fair policy had the lowest data reception average at 60%, this was while it managed to accommodate the high traffic bursts of Deluge and allow 100% dissemination success. Greedy Queues managed 75% data reception, but with a large number of collisions, and the complete failure of Deluge.

These results show us that it is possible to create a scheduling layer using only local information to enable multiple protocols to function reliably at data rates where there had previously been protocol failure because of high data rates. The next set of experiments examined whether this same simple scheme could work recursively for multi-hop networks in an epidemic like fashion.

4.9 Multi-Hop Evaluation

My scheduling policies were designed to optimise local, single hop communication. I wanted to see their effect on multi-hop communication scenarios. For these experiments I decided to only evaluate the scheduling policies against the default TinyOS round robin scheduler. The Fair

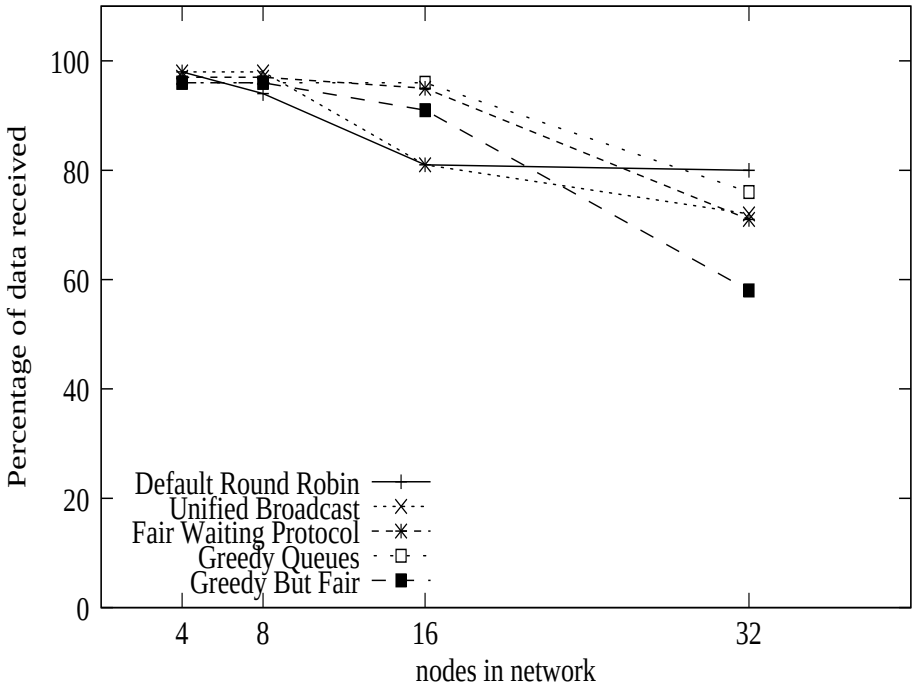

Figure 4.14: Plot of the percentage of data message delivered using CTP during the Deluge dissemination.

	dissemination time	avg packets collected
Standard	100% (431.5 seconds)	80%
Greedy Queues	100% (242.67 seconds)	84%
Greedy But Fair	100% (252.3 seconds)	82%

Table 4.3: The average percentage of data packets received by CTP

Waiting Protocol only adds delays to reduce contention, and in a multi-hop environment I found that the delays compounded to make the system performance very slow. The Unified Broadcast network layer was excluded from multi-hop evaluation because the latency introduced through message buffering caused difficult to debug timing errors when a single node in the network could receive messages from two other nodes hidden from each other (hidden terminals) for the FTSP synchronisation protocol. Another reason was that UB increased the message payload size from 28 to 60 bytes, therefore making the comparison one about optimal message sizes and not about scheduling policies.

4.9.1 In Lab Testbed

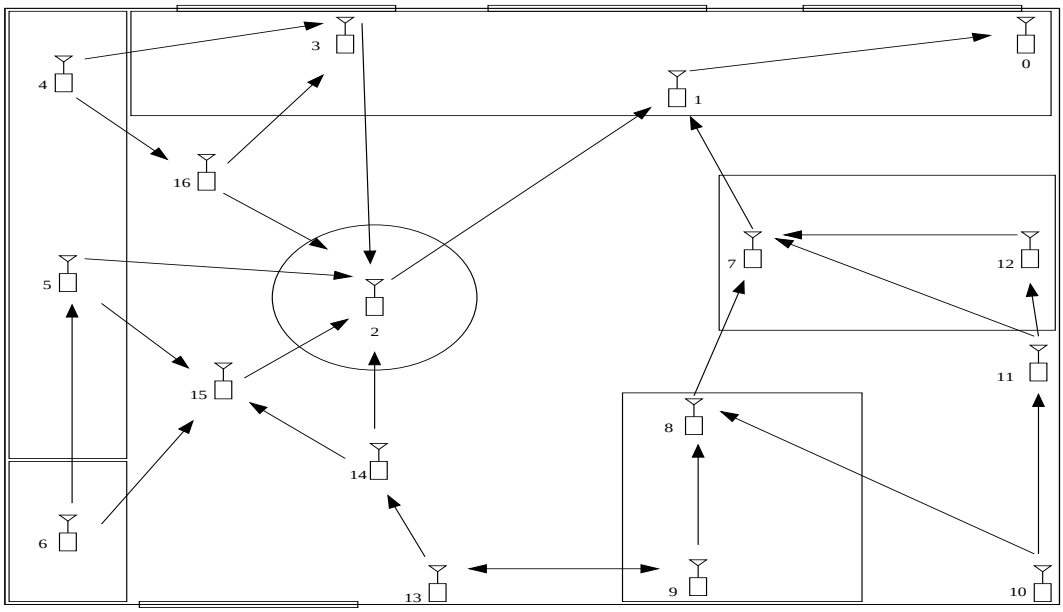

Figure 4.15: Topology of the network used in the multi-hop experiments. Each sensor node has an id number.

The first multi-hop experiments used the same experimental setup that were used for the one hop networks with 16 nodes and one base-station (see Figure 4.15). The radio transmission power was reduced to power level 5 on the CC2420 radio used by the MicaZ. This produced -20dBm of output power and created a network with a maximum hop depth of 4 hops to the collection base-station. See Figure 4.2 for a table of the parameters used.

CTP was used for data collection, and the nodes sent data at the rate of one data message every eighth of a second. This rate was chosen by experimentation to try and be as close to the edge of the capacity region of the network as possible. A separate Deluge base-station was used to inject and disseminate new code images into the network because it is difficult to have one base-station handle both dissemination and collection. Each code image was 38 pages, and each page was 1024 bytes. The time was measured from when the deluge command was issued, to the time at which the last node began its reboot. The results are shown in Table 4.3.

It can be seen from these results that the average time to disseminate is roughly 43% faster with both the Greedy Queues and Greedy But Fair schedulers than with the default TinyOS round robin scheduler. Data collection is also improved by 2-4%. In all cases the FTSP synchronisation protocol functioned properly. This experiment shows that although the Greedy Queue scheduler only uses local information to determine the protocol to send and the backoff

delay, it still provides benefits over the standard scheduler in a multi-hop network.

4.9.2 Remote Testbed

The next set of experiments deployed the same set of protocols on the remote wireless sensor testbed Indriya [DCA12]. The testbed consisted of 138 functioning nodes at the time these experiments were run, with a maximum depth of 8 hops. This facility allowed us to evaluate the protocol in a large multi-hop environment, where the population of the nodes is fixed. To vary load I increased the sending rate of CTP, and then periodically disseminated large amounts of data using a Polite broadcast dissemination protocol similar to Deluge. I measured the percentage of data packets received at the CTP base-station, time in seconds to disseminate 30Kbytes to each node in the network, and the variation of the timestamps on the data packets received at the base-station.

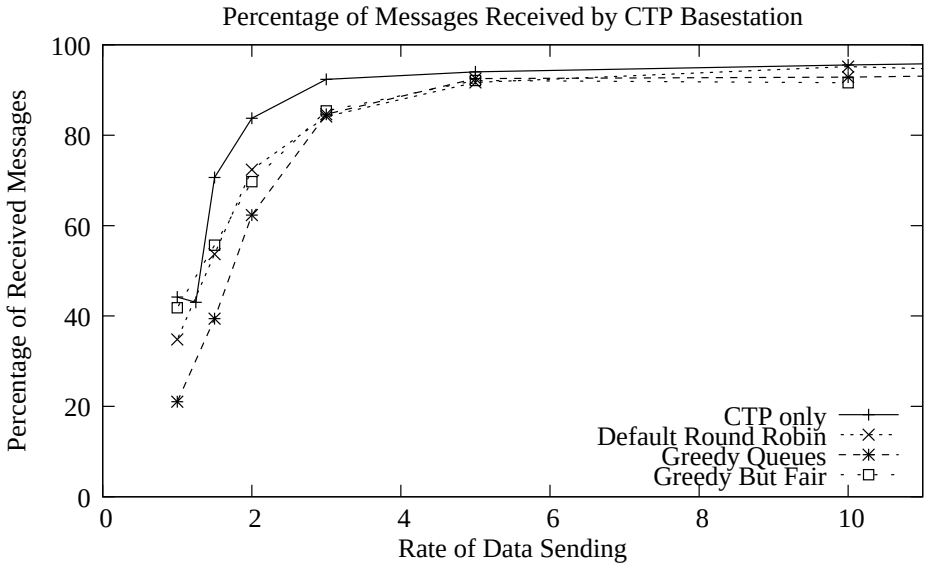

Figure 4.16: Percentage of data packets received by CTP per data sending rate.

4.9.3 Multi-Hop Evaluation

Figure 4.16 shows that as the rate of data sent by each node via CTP increases, the percentage received by the base-station diminishes. When CTP is on its own, not sharing the network

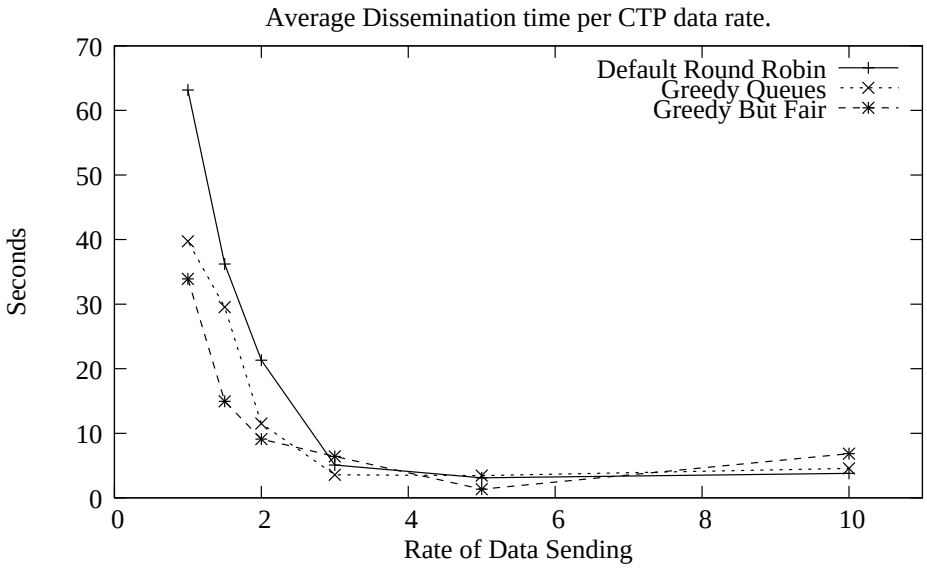

Figure 4.17: Average time to disseminate new data to network using Polite Broadcast per CTP data sending rate.

with other protocols, its performance begins to degrade at a send every two seconds. At a send every second, the base-station receives less than 45% of the data messages sent.

The Greedy Queues scheduler, the Greedy But Fair scheduler, and the round robin TinyOS scheduler have very similar performance to CTP on its own up to the rate of one message every five seconds. At a message every three seconds, all the scheduling policies lose the same percentage of data messages. At higher data rates, the Greedy Queues scheduler loses on average 5% more messages that either the default round robin or Greedy But Fair schedulers. This pattern continues all the way to one message every second. At this point the Greedy But Fair scheduler performs slightly better than round robin.

The results in Figure 4.17 show that under a heavy data communication load, the Greedy Queue scheduler reduces dissemination time. The times are very similar at lower CTP data rates. At one message every three seconds the dissemination messages start to get delayed. At one data message every two seconds the Greedy Queues dissemination delay has doubled. The Greedy But Fair scheduler's delay has only increased by about 50%. The default Round Robin scheduler sees its delay increase by almost four times the delay experienced at one data message every two seconds. By the CTP data rate of one message every second, the Greedy Queue scheduler is 35% percent faster than the default Round Robin scheduler. The Greedy

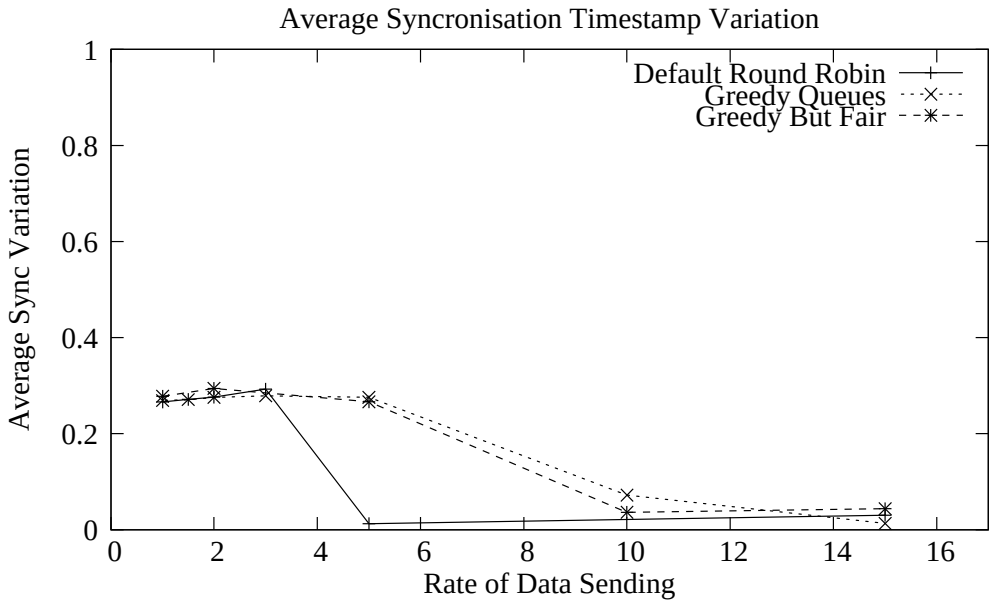

Figure 4.18: Average variation of timestamps received at the base-station per data sending rate.

But Fair scheduler has roughly one half the dissemination delay of the default Round Robin scheduler.

The most interesting result of the large, multi-hop remote WSN test-bed is that the FTSP time-synchronisation protocol fails at higher data rates. These results can be seen in Figure 4.18. With a data rate of one message every 15 seconds to one message every ten seconds FTSP works well, and all of sensor nodes in the network are taking synchronised samples. When the data sending rate is increased to one message every five seconds, FTSP fails for both of the Greedy schedulers. FTSP continues to work for the default round robin TinyOS scheduler. When the data sending rate increases to one message every three seconds, FTSP fails on the round robin scheduler as well.

FTSP fails for both of the Greedy scheduling approaches because, as the data rate increases and disseminations occur, the infrequent time-synchronisation packets sent by the FTSP root node get queued and delayed. This latency causes FTSP to fail, and be unable to synchronise the sensor nodes.

The Multi-hop evaluation results are similar to those seen in the single-hop evaluation. The percentage of data packets received at high data rates are similar to those of the round robin

scheduler with the Greedy But Fair policy, and only slightly worse with the Greedy Queue scheduler. The dissemination time for both Greedy Queue schedulers is better than that of the default round robin scheduler. The most interesting difference is that in large, multi-hop networks, the FTSP synchronisation packets were delayed in the queues causing FTSP to fail.

4.10 Greedy Queue scheduling Conclusions

WSNs provide the ability to detect spacio-temporal phenomenon hitherto difficult or impossible to obtain, like the movement of smoke in a room, or the distribution of moisture in a field. The current practice of creating WSN systems is to combine several different protocols to provide services such as data collection, synchronisation, and information dissemination. It was shown in this chapter that this approach is problematic and that the communication requirements of one protocol can interfere with the functioning of another protocol. This situation presents to us a problem with the way WSN systems are formed, and illustrates the WSN management problem. Services need to be provided, while communication needs to be kept to a minimum to conserve energy.

In this chapter I experiment with the use of a biological-style distributed computation to solve the problem of inter-protocol interference of service protocols. I wanted to use existing WSN service protocols, and prevent them from interfering with one-another. I created a distributed scheduler where each node used only the information of its local neighbours to schedule its many protocols, and its access to the communication medium. The limits that I found was that it is impossible to use a scheduling approach once the network capacity has been exceeded, and that latency would cause the synchronisation protocol to fail in a large multi-hop network.

The results which I show in this chapter suggest that by using the same style of local-only information as seen in biological computation, my distributed queue-based scheduling policy can enable multiple protocols to function together. I observed a 35% decrease in dissemination time when using the Greedy Queues scheduler, and a 48% decrease for the Greedy But Fair scheduler. This result is valuable because it indicates the need to find a more coordinated way

to manage the communication overheads of service protocols than is currently being used in WSN systems. Communication management, such as the Greedy Schedulers, will be needed in order to provide all of the services needed by the smoke monitoring application, or any other environmental monitoring application. A WSN operating system cannot export a radio abstraction which allows any protocol to believe it has a private radio, and function in ignorance of the other services running on a node. There needs to be integration at the protocol layer, so that all of the service protocols work in a common way so that they can be scheduled together, and made to work efficiently.

My schedulers were designed and analysed for a single-hop network. I demonstrated that they also work well in a multi-hop environment. A problem which I encountered in a large multi-hop network was that the queues used by the schedulers can cause a delay in the sending of a message. If the protocol is sensitive to message delay, like the FTSP synchronisation protocol, then the protocol will fail.

An interesting result that I found in my experiments was that simple message aggregation, such as that used by Unified Broadcast, is very effective in reducing communication overheads. I also saw that the message aggregation approach had limits and negatively affected the performance of the service protocols. I will further investigate this approach in later chapters.

The results in this chapter provide the first piece of evidence to support my hypothesis that biological-style computation can be used to enable the self-organisation and self-management of resource constrained WSN nodes. I showed that using only local information my schedulers were able to provide good performance within the network capacity region. The Greedy But Fair scheduler also enabled the dissemination protocol to continue to function beyond the capacity region, when the other schedulers had failed.

In the next chapter I explore the use of another approach towards using biological computation to enable node self-management. Instead of trying to use pre-existing protocols, I develop my own. I explore the use of a bio-inspired swarm-intelligence approach to offer more communication efficient services. The first service I implement is a time synchronisation protocol. That is because the Greedy Queue schedulers can cause message delays that would cause synchronisa-

tion protocols to fail in large multi-hop networks. Time synchronisation is both a core service to environmental applications which collect spatio-temporal data, and to other WSN system services which function in a synchronised fashion.

Chapter 5

Synchronisation

5.1 Introduction

Synchronisation is the core service for the smoke monitoring example application and for all environmental sensing applications. It enables all of the nodes in the network to take air quality samples at the same time. The location of smoke can be tracked as it moves around the room, and the samples will be correlated in time, and located in space. Without this service, it would be impossible to track the movement of the air pollutants, and the data would not have any value to the application.

In the previous chapter, I provided the first piece of evidence in support of my hypothesis that biological computation realized through bio-inspired protocols can be used to enable wireless sensor nodes to self-manage communication bandwidth in an efficient, and throughput optimal, way. In this chapter I provide more evidence in support of this argument through the provision of the core WSN service of synchronisation using a bio-inspired, swarm intelligence algorithm. I aim to make this service as communication efficient as possible. In the next chapter I will show how it can work well with other bio-inspired service protocols, solving the same problem of multiple protocols without requiring a scheduler.

Current solutions to WSN synchronisation have high communication overheads, are reliant on

a single point of failure, or do not integrate well with the other service protocols being used by the WSN. I solve these problems with a completely decentralised synchronisation mechanism based on epidemic propagation (I use the terms epidemic and gossip as synonyms) which allows the sensor nodes to synchronise themselves at the network level, and can provide time correlated event notification to a user application [COKSM05]. I do this with low communication overheads. The decentralised nature of the protocol makes it immune to single points of failure.

There are two types of synchronisation: global (absolute), or event (relative). Global or absolute synchronisation ensures that all nodes have a clock that gives the same value for the same point in time. This value can be expressed in seconds, milliseconds, or any other agreed upon time unit. It is the equivalent of making everybody in a room set their watch to the same exact time, so that if you asked the time, everybody would shout the same response in unison. Event or relative synchronisation only ensures that everyone act at the same time, there is no global value of time. This is identical to the case of having everybody in the room clap together at the same time. The claps would happen in unison, but the value on everyone's watch will probably be different.

A common use of event synchronisation in WSNs can be seen in duty cycling. Applications with long periods of time between samples, such as soil moisture monitoring [COKSM05], can duty cycle (turn off during periods of inactivity) the sensor nodes to save energy. Duty cycling depends upon the node's ability to wake themselves at the same time to take meaningful samples, and to be able to communicate with the rest of the network. Waking simultaneously becomes difficult over time because of the clock drift inherent in the cheap oscillators used in wireless sensors [GGS+05]. Synchronisation is needed to re-sync the clocks every waking period, to compensate for the clock drift from the previous sleep period. Synchronisation is central to the ability to duty cycle, and therefore save energy.

There are several key requirements for a WSN synchronisation protocol. The first is robustness to node failure. Failure is common and frequent in WSN. Nodes can be deployed in difficult environments such as potato fields [LBV06], glaciers [MOH04], or the backs of zebras [LSZM04]. The conditions of these environments can increase the rate of node failure due to moisture,

temperature effects on the batteries, or external intervention by children or animals. The second is that the protocol should be scalable. This is another general requirement for WSN as sensor node populations can become large for some applications. The final requirement is embodied by the WSN management problem. Synchronisation is a system level service, and needs to be provided at a low communication cost along with other system services such as routing and dissemination.

In this chapter I present a bio-inspired solution to WSN synchronisation which I call the Epidemic Synchronisation Protocol (ESP). First I present the state of the art in WSN synchronisation protocols and identify their weaknesses. I then describe the synchronisation protocol ESP. An implementation of the protocol is described and evaluation results are then presented and discussed.

5.2 State of the Art

Several protocols exist for WSN synchronisation [SBK05]. All synchronisation protocols use the concept of time periods. A time period (or period) is a constant, non-overlapping unit of time used by all the nodes in a network. For example, a period could be four seconds (see Figure 5.1). All of the sensor node's periods start at the same time when all of the nodes in a network are synchronised. When the time periods of all of the nodes start in synchronisation, they are in phase. The use of periods allows all of the nodes to perform the same action at the same time. This allows the nodes to function like a coherent system. For example, all of the nodes can agree to sample their sensors at the middle of every period. That way all of the sensor data can be correlated in time.

The most serious problems shared by the current WSN synchronisation protocols are high communication overheads. This occurs because they require that every node broadcast every period. Those that do not require every node to broadcast every period use central synchronisation nodes, and are therefore susceptible to a single point of failure. Another problem with many of the current synchronisation protocols in use is that they are independent protocols

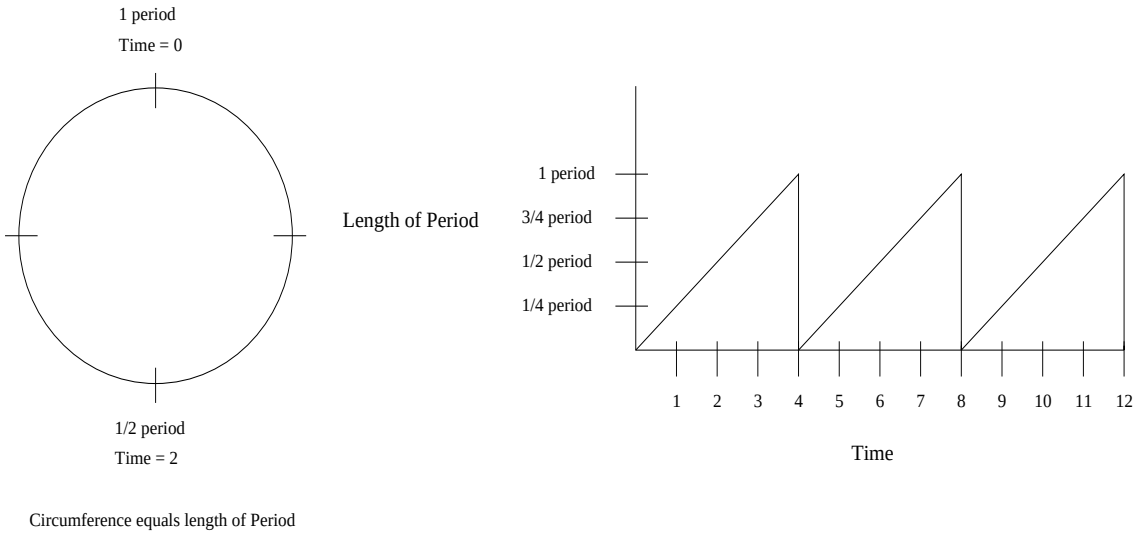

Figure 5.1: A graphical representation of a time period. Note that each period has a duration of four time units.

and may not cooperate with the other service protocols used by the WSN.

The state of the art WSN time synchronisation protocol is the Flooding Time Synchronisation Protocol (FTSP) [MKSL04]. FTSP failed in the previous chapter when there was a high communication traffic load in a large, multi-hop WSN testbed. This protocol chooses a random node (the one with the lowest ID) and uses that node as the root node for synchronisation. The root node broadcasts a sync message with its time-stamp. The time-stamp is written and read at the MAC layer. MAC layer time-stamping reduces system introduced delays that would make the synchronisation protocol less accurate. All of its one-hop neighbours receive the broadcasts and store them. When a sufficient number of sync messages have been received by a node it calculates its offset and skew using a linear regression of all of the time differences between their time-stamp and that of the root node. In this way all of the neighbours of the root node can use their calculated offset and skew, and produce the same time-stamp as the root node. The first hop neighbours of the root node then act as root nodes for all of the nodes two hops away from the root node. The time-stamp offsets are then flooded around the network, with each node synchronising itself in a recursive fashion.

By using a single root node and linear regression, FTSP is able to synchronise an entire network with a very high accuracy. The weakness of this protocol is its dependence on a single root. Loss of that root, or the inability to receive the synchronisation messages, will force the network,

or parts of the network, to perform re-synchronisation with a new root. This single node dependence also means that multi-hop (in the form of flooding) control messages need to be propagated, adding communication traffic to the network. In the last chapter it was shown that when high communication traffic in a large multi-hop WSN caused message delays, FTSP failed.

The Gradient Time Synchronisation protocol (GTSP) focuses on maintaining a low synchronisation error between two immediate (one-hop) neighbours. It also maintains synchronisation between neighbours which are multiple hops away, but with a larger synchronisation error. It works by having every node periodically broadcast a synchronisation message to its neighbours. The message contains the current time and the clock rate. The sync beacons are stored by the receiving nodes in a neighbour table. A node calculates its clock rate for the next period by finding the average of all of its neighbours clock rates. It then sets its current time value to the average of all of its neighbour time values. When this algorithm converges, all of the nodes in a network agree on the same clock rate, and the same global time value.

The authors of GTSP reported a one-hop synch error of 4 μseconds. The drawbacks of this approach are the heavy communication overhead with the requirement that all nodes broadcast once per period. It also has a high memory requirement, each node having to store the current time and clock rates of all of its neighbours.

A very similar algorithm is the Average Time Sync (ATS) protocol. It calculates the average clock rate from those of local area neighbours. In addition, it calculates the average clock offset to compensate for clock drift. This protocol improves upon GTSP by not requiring the use of neighbour tables. Instead, it calculates its average values with every synchronisation message received. This protocol still fails to overcome the problem of high communication costs of GTSP. It requires every node to broadcast every synchronisation period.

The Gossip Time Protocol (GTP) [IVSV06] is very similar to FTSP. It depends upon a root node to which all of the other nodes in the network synchronise their clocks. It uses a unicast push-pull gossip (epidemic) protocol to disseminate the time-stamp. It was not designed for use on WSN, so it does not take into account the problems of communication overhead and

unreliable messages inherent to WSN. GTP would not work very well on WSN because it would have a high communication overhead, and it would have a hard time dealing with the unreliable communication patterns of WSNs.

Mathematical Biology has produced a model of phase synchronisation based on the synchronous flashing of Malaysian fireflies [Buc88] referred to as Pulse Coupled Oscillators. Mirollo and Strogatz proved that a fully connected network of pulse coupled oscillators is guaranteed to converge to a state of synchrony [MS90]. This model has received a lot of attention in the WSN community as the basis of many decentralised synchronisation algorithms [WLP+02, WM04, PS07, WATP+05, YM11].

The essence of the pulse coupled oscillator algorithm is that all of the nodes listen to when their neighbours fire. When they hear a neighbour fire, they reduce the time of their next (and only their next) firing to bring their firing period into phase with their neighbour's. The aim is that, over time, all neighbours will fire at the same time. Several WSN synchronisation protocols have been derived from the pulse coupled oscillators model. Here I discuss three: the Reach-back Firefly Algorithm (RFA) [WATP+05]; the Meshed Emergent Firefly Synchronisation protocol (MEMFIS) [TAB10]; and the Pulse Coupled Oscillator Protocol [PS07].

RFA is a phase synchronisation protocol for event synchronisation. It assumes that all of the nodes have the same event frequency, but start out of phase. An event refers to some action such as taking a sensor sample or flashing an LED. Every node performs an event once a period. All of the periods are the same length, τ seconds. Each node has a clock which counts from 0 to τ. At time 0 the event occurs, and then the clock counts to time τ before another event is triggered again. At time τ, the clock is reset to 0. In the time between events, the node observes its neighbour nodes. Whenever a node hears an neighbour node's event, it records the time. After the nodes own next event occurs, the node calculates the offsets from the times of the previous neighbour events it has seen. If the sum of the offsets is less than its period, it reduces its next period length by the offset. This causes the node to fire sooner, more in sync with its neighbour nodes. If the calculated offset is greater that the period, it does nothing. See 5.1 for details. After several event periods, all of the nodes will fire their events at the same

time, and have synchronised their periods.

$$t_1 = \begin{cases} t_0 + \epsilon * \Delta t & \text{if } time_0 + \epsilon * \Delta t < 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

The ϵ in the above algorithm refers to the percentage of the Δt used to make the local adjustment. Say that the period of the clocks is 100 seconds ($\tau = 100$), and two nodes are out of sync by 10 seconds. If the epsilon is .1 (10%), then for the next period the receiving node will reduce its period by $\epsilon * \Delta t$ seconds, or one second.

The RFA protocol has two major problems. The first is that it is slow to converge. The experimental results presented in [WATP+05] show a best case synchronisation time of four and a half minutes. The other problem with RFA is its high communication overhead. It requires that every node broadcast when it fires every period.

The Meshed Emergent Firefly Synchronisation protocol (MEMFIS) is a synchronisation protocol based on the same Pulse Coupled Oscillator model as RFA [TAB10]. This protocol is designed to align slotted communication at the MAC layer. It reduces overhead costs by combining the synchronisation information into data communication, but still requires that every node communicate. MEMFIS uses a custom built transceiver which multiplexes communication data with synchronisation data. The reliance upon custom hardware does not make this protocol very general.

PCO [PS07] also uses the Pulse Coupled Oscillator algorithm. It explores the use of refractory periods to decrease the time required for nodes to synchronise. A refractory period is a portion of the time period where it will not adjust itself to a neighbour node's synchronisation message. The use of the refractory period is to prevent susceptibility to radio phenomenon such as reflection and ghosting which will cause the reception of a copy synchronisation message. As the nodes adjust to a message based on time of arrival, if a second copy of a message is received after a node has fired, then it will adjust to this message. If the message is received after a nodes firing, it will be a long way out-of-phase with the receiving node, and make a large adjustment.

This large adjustment to an erroneous message will make the node unable to synchronise. The refractory period was suggested in [HS05] and further discussed in [DBR08]. If a node has a period of 1000 milliseconds, then a refractory period of 50% means that all firing messages received in the first 500ms are ignored. Figure 5.2 is a diagram showing the use of a refractory period equal to half the total synchronisation period.

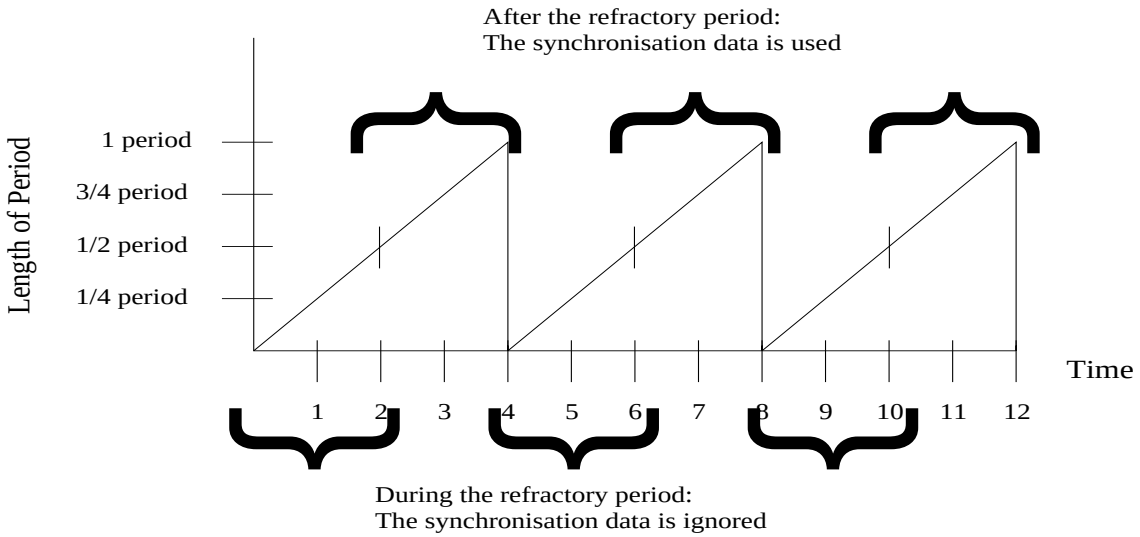

Figure 5.2: Diagram explaining the way data from a message is used if it is received during or after the refractory period. A message received during the refractory period has its sync information disregarded. After the refractory period, all information is regarded.

PCO [PS07] was designed for ultra wideband radio, and only used the time of message arrival to synchronise the network. Each node would broadcast a signal every period. The signals would overlap as the nodes synchronised. As the time of arrival of the message was important, not the data contained in the message, the signal would increase as more nodes became synchronised. This protocol had a much shorter synchronisation time than RFA. Its reliance upon special radios for synchronisation prevents its use as a general synchronisation protocol. This protocol also required the communication of every node every synchronisation period.

A different approach to distributed synchronisation is given in [YT08]. This method uses randomised circular averaging algorithms, and compares itself to the original RFA [WATP$^+$05]. This approach presents a different approach to decentralised synchronisation. Evaluation in [YT08] shows that circular averaging algorithms are robust to node and message loss in synchronised networks. The problem with them is that they do not offer very accurate synchro-

nisation for environmental monitoring applications (with synchronisation errors in seconds, as opposed to milliseconds for other schemes), and take a long time to converge.

As can be seen from the examples given above, the problems of either high communication overheads or the reliance of a single synchronisation point are common to all WSN synchronisation protocols. I solve the problems mentioned above with a completely decentralised synchronisation mechanism based on epidemic propagation. With ESP, the nodes synchronise themselves and provide a time correlated event notification. I also examine the amount of information required by this protocol to produce the desired state of synchronisation, thus addressing the WSN management problem.

5.3 Epidemic Algorithms

The bio-inspired, swarm-intelligence algorithm which I use heavily in this work is Epidemic Dissemination, also referred to as Gossip (note: I use the terms epidemic and gossip interchangeably). Epidemic algorithms were first proposed as a simple, decentralised way to disseminate updates in databases [DGH$^+$87]. This class of algorithm is based on a model of how diseases spread in a population of potential hosts. It is also referred to as Gossip [Jel11], and is used as a model for the way rumours are spread.

Epidemic algorithms are a good example of bio-inspired swarm intelligence algorithms which show the behaviour of a class one cellular automata (where all of the nodes converge to the same state). A good example of this is gossip based network computation [JMB05]. In this work, the ability of gossip-based protocols to be used for more than just information dissemination is discussed. Examples are given of algorithms which can calculate a global average, maximum, minimum, or median value for a given parameter on the nodes in a WSN. This allows us to compute a global function over all of the nodes, by combining communication and processing. This is a way of realising biological computation.

The notion of biological computation extends to the use of gossip-based algorithms to provide self-adapting properties to distributed networks [BJ08]. Giving a network of nodes the ability

to calculate average, maximum and minimum local and network values allows the sensor nodes to self-adapt to improve their performance. One example of this is allowing a network of nodes to choose a single time-stamp value, say the largest one, and then globally adapt to it. This removes the need for a single node to provide a time stamp for global use, and therefore removes a possible single point of failure.

5.3.1 How Epidemic Algorithms/Gossip Work

Gossip works by having each node in a network keep a list of local neighbours [DGH+87, Jel11]. The nodes run two processes, one is passive and listens for another node wanting to 'gossip' (figure 5.3). The other is an active process which picks a random neighbour from its neighbour list (line 3, figure 5.4), and sends that neighbour a list of its most current information. If the node and its neighbour both have the same version of information, then the exchange is done, and no more communication takes place. If one of the nodes discovers that the information it possesses is old, then the node with the older information requests the newer information from the other node. The newer information is then transferred, and both nodes sleep again until the next round.

Take, for example, a network of WSN nodes. New data needs to be disseminated to all of the nodes in the network. The data is marked by a version number. This is a monotonically increasing value which records the age of the data. A higher version number indicates newer data. A node with new data and therefore a higher version number randomly chooses one neighbour with which to exchange its version number and new data. If the receiving node finds that it has the same version and data as the sending node, then nothing more happens. If the receiving node discovers that it's version is less, and therefore older than the senders it changes its version and data to the new ones. It then chooses one of its neighbours to randomly send data to. If a node receives a smaller, and older version number than its own, then it immediately sends its newer information to update its neighbour. Epidemic algorithms solve the broadcast storm problem (if every node in a network repeats a message it hears in order to forward it, then the message usage increases exponentially, and the network suffers congestion)

by reducing the amount of communication used to update the network.

There are many different forms and variations of epidemic dissemination, one of the most fundamental being whether the algorithm pushes the data, pulls the data, or does both. Pushing is the example given above, where the node with the new data initiates communication. Pulling data is where a node requests new information, usually by periodically polling the network. Push and pull algorithms combine the two approaches.

The fundamental primitive behind Gossip is the use of randomisation. During the passive thread, a neighbour is chosen at random for communication. The fact that the neighbour choice is random enables the protocol to disseminate information efficiently, when compared to broadcast flooding, while remaining very robust to random node failures.

```
1: loop
2:    Listen for peer info_θ from θ
3:    Receive peer info_θ from θ
4:    Send peer θ local info
5:    if peer info_θ is newer than local info then
6:       download peer info_θ
7:       replace local info with peer info_θ
8:    else if peer info_θ is older than local info then
9:       send local info to θ
10:   else
11:      do nothing
12:   end if
13: end loop
```

Figure 5.3: Algorithm for passive process of gossip protocol.

5.4 The Epidemic Synchronisation Protocol

My decentralised WSN time synchronisation protocol is called the Epidemic Synchronisation Protocol (ESP) and is presented in pseudo-code in Figure 5.5, and the way timing affects the way messages are interpreted is shown in Figure 5.2. It is based on the idea of using epidemic algorithms to synchronise the WSN nodes in a distributed way [JMB05, KDG03]. ESP synchronises both events and creates a global time-stamp.

```
1: loop
2:    Once every random time t in time period τ
3:    Choose random peer θ from peer list
4:    Send peer θ local info
5:    Receive peer info_θ from θ
6:    if peer info_θ is newer than local info then
7:       download peer info_θ
8:       replace local info with peer info_θ
9:    else if peer info_θ is older than local info then
10:      send local info to θ
11:   else
12:      do nothing
13:   end if
14: end loop
```

Figure 5.4: Algorithm for active process of gossip protocol.

```
local_clock = 0
cycle_length = 100
transmit_local_clock = TRUE
loop
  if local_clock == cycle_length then
    if transmit_local_clock == TRUE then
      transmit local_clock and local_metadata value
    else if transmit_local_clock == FALSE then
      NO TRANSMISSION
    end if
    local_clock = 0
  end if
  if clock ≤ cycle_length then
    listen
    if A message is overheard AND local_clock < refractory_period then
      IGNORE MESSAGE
    else if A message is overheard AND local_clock > refractory_period then
      adjust local_clock to that of received neighbour
      transmit_local_clock = FALSE
      if The message contains timestamp > local_timestamp then
        local_timestamp = timestamp
      else if the message contains timestamp < local_timestamp  then
        transmit local_timestamp now
      end if
    end if
  end if
  local_clock = local_clock + 1
  global_timestamp = local_timestamp * cycle_length
end loop
```

Figure 5.5: Pseudo code algorithm for the dissemination of timestamps with epidemic synchronisation

ESP differs from Pulse Coupled Oscillator based protocols in three important ways. The first is that it uses message suppression to limit the number of synchronisation messages each node will hear. This allows us to greatly reduce the communication overhead required by ESP. The second difference is a direct result of the first. Because ESP reduces the number of nodes sending synchronisation messages, it does not adapt its firing time to a portion of the received firing time, referred to in the literature as a coupling factor. ESP completely changes the next period so that a node fires at the same time as the node which sent the synchronisation message. This method treats the firing time as information to propagate. The final important difference is that ESP is able to synchronise events and provide a global time-stamp. PCO based protocols provide event synchronisation, ESP can also provide a unified global time-stamp.

I evaluate the ability of ESP to converge, and the stability of that convergence experimentally, both through simulation and in test-bed experiments.

5.4.1 Broadcast Gossip and Message Suppression.

The first problem ESP addresses is the high communication costs of other decentralised synchronisation protocols in terms having every node broadcast every period. I start by looking at centralised synchronisation protocols such as FTSP. It uses a single central node to act as a main node to which all other nodes will synchronise to. Centralised protocols tend to be faster to converge and have a lower communication overhead. A way of approximating this behaviour is to use a leader election protocol to chose local 'central' nodes and therefore have a hierarchical structure such as that used in LEACH [HCB00b]. Next, I explore ways to reduce the communication needed without using a central sync node or a local election process.

To reduce the communication overhead required in PCO synchronisation Degesys et al. [DBR08] experimented with probabilistically reducing the number of sync messages. A pseudo random number generator was used with a fixed probability to determine if the next sync message was to be sent or suppressed. The probability used in [DBR08] was 0.2 which suppressed 80% of the synchronisation messages transmitted by the network overall. It was found that the time to sync was not greatly affected, and the percentage of nodes reaching a state of synchronisation

was also unaffected. Several topologies were investigated: ring, line, random. The performance of other probabilities ($p \neq 0.2$) was not investigated in [DBR08]. Random message suppression succeeded in reducing message overhead to 20% of that of the original RFA. The results in this work were based solely on numeric simulations and therefore do not take into consideration radio effects, or any system latencies that would have been apparent had a packet based simulator like TOSSIM or an implementation on real sensor nodes been used.

In order to further reduce the communication used I decided to base ESP on a polite gossip protocol. In the standard gossip protocol two nodes are chosen at random to communicate. The two nodes then exchange information with one-another. I use a broadcast form of gossip referred to as polite gossip [L+03]. In this form of gossip, every node sets a random timer at the beginning of each synchronisation period. When that timer fires, the node broadcasts the value of its local state variable. If a node receives the local state variable of another node before it fires, it compares that to its own. If the received variable has the same value as its local variable, then the receiving node cancels its own broadcast.

The conditions under which ESP will suppress a synchronisation broadcast is different to the conditions under which Degesys et al. will suppress broadcasts. Nodes using ESP suppress synchronisation broadcasts when they have received a certain number of neighbour synchronisation messages, after the end of the refractory period. The protocol presented by Degesys et al. cancels broadcasts randomly, based on the results of a locally generated random number. The suppression occurs regardless of the reception of any neighbour synchronisation broadcasts. These two approaches will be compared in the evaluation of ESP.

Polite gossip is used in a WSN information dissemination protocol called Trickle [L+03, LBC+08]. This variation of gossip assumes broadcast communication and works by using a variable communication window. A long window time is used when there is no new data to propagate in the network. As soon as new data is received by a node, it reduces its communication window to increase the rate at which it informs its neighbours that new information is available. By using a long time window when there are no updates present in the network, and a short window when an update is detected Trickle tries to balance message use with dissemination time.

With Trickle, nodes choose a random time to transmit the version number of their data. The remainder of the window, nodes just listen for the broadcasts of other nodes. If a broadcast contains the same code version as that of a listening node, then the listening node remains silent, and deschedules its next broadcast. If a node hears the broadcast of a code version more recent than its own, then it responds to the broadcast with a request. The request can be filled by any node that has the newer version of the code, not just the original broadcaster. If a node is listening and hears a node broadcast a code version older than the one it has, then it immediately broadcasts the fact it has newer code, followed by the new code itself.

A broadcast form of epidemic communication suits WSNs. It only requires sending, and no acknowledgements as with traditional epidemic algorithms, and so it has a lower communication overhead. Another reason is that WSN nodes communicate using radio. This form of communication is inherently broadcast. It is best to leverage this fact and try to do as much communication as possible with each transmission.

ESP uses a form of polite gossip to synchronise but, it does not use Trickle. The variable listening window used by Trickle would constitute a variable period length, and I assume that all nodes have the same period. ESP also alters the meaning of different parts of a period by using a refractory period. During a node's period, it receives synchronisation messages from its neighbours. If a message is received in the first half of the period, it is ignored. This period is referred to as the refractory period. If a message is received in the second half of its period, it will change its next period length so that it will be in phase with the sender. The node will then cancel the sending of its synchronisation message for the next period.

5.4.2 Firing Time Adjustment.

ESP's use of polite gossip for synchronisation means that the nodes will synchronise to a small number of neighbours each period. This is different to the function used by other Pulse Coupled Oscillator algorithms which synchronise to all of their neighbours each period. ESP has a lower communication overhead as a result.

A node using a Pulse Coupled Oscillator protocol like RFA, MEMFIS, and PCO will change the length of its next period based on the firing times of all of its neighbours. Because the node uses such a large sample size (all of its neighbours), that change uses only a percentage of the difference between the start of its period and the start of its neighbour's periods. This is shown in Equation 5.1. The ϵ in the 5.1 refers to the percentage of the difference in period start times (Δt) used to make the local adjustment. For instance, if the period of the clocks is 100 seconds ($\tau = 100$), and two nodes are out of sync by 10 seconds and epsilon is .1 (10%), then for the next period the receiving node will reduce its period by $\epsilon * \Delta t$ seconds, or one second. This allowed a node to sum up the differences from all of its neighbours.

In general, nodes using ESP synchronise to only one neighbour. There is no need to aggregate a bunch of small differences. In multi-hop networks there is the chance that nodes will hear multiple synchronisation messages, because the sending nodes can not hear each other and will not cancel their synchronisation messages. This is known as the hidden-terminal problem. A hidden terminal in a multi-hop network can be simply defined. Given a network N of $|N|$ nodes numbered $1 \ldots |N|$ an individual node is referred to as $n_i \in N$. For any (and all) of the nodes in N, say for example $n_1 \in N$, all of the other nodes $n_2 \ldots n_{|N|} \in N$ will be one or more hops from n_1. Any node node which is more than one hop from n_1 is a potential hidden terminal. Figure 5.6 shows a graphical representation of the hidden terminal where there are three nodes: A, B, and C. In this image A can hear both B and C. B and C are hidden from each other. The ESP algorithm will change its phase to the average of the two differences upon the reception of multiple synchronisation messages from nodes hidden to each other. This algorithm can be seen in 5.2.

$$t_1 = t_0 - \overline{\Delta t} \tag{5.2}$$

From a biological computation point of view this protocol performs two different computations. The first is to get all of the nodes to converge in time. This allows all of the nodes to perform the same event in synchronisation. This is done assuming that all of the nodes have the same event frequency, but are out of phase. I describe the second biological computation below.

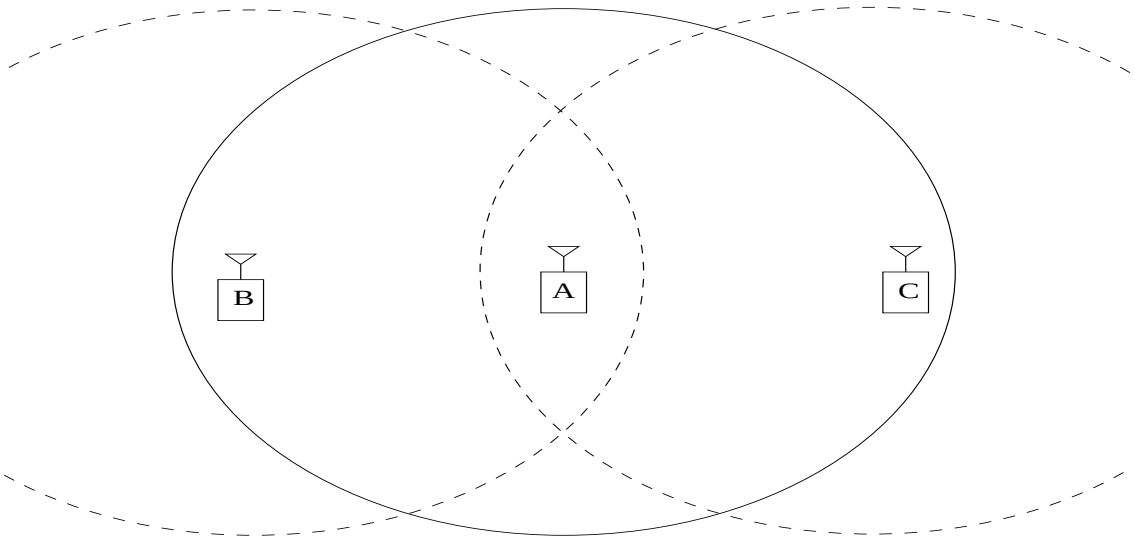

Figure 5.6: Graphical representation of the hidden terminal problem. Of the three nodes shown, A can hear B and C, and B and C can hear A. Nodes B and C are hidden from each other since they are more than one hop away from each other.

5.4.3 Global time-stamps

Once the network is synchronised to perform the same event at the same time, the next step is to create a global time stamp. This allows all of the nodes to sample at the same time, and then assign a value to the sample which would signify which samples were taken at the same time. This is the result of biological computation which outputs an arbitrary, global time stamp.

In the case of FTSP, there is an arbitrary central node which disseminates its time, and all of the other nodes calculate their offset from this time locally. The time-stamp value is arbitrary.

I can exploit the fact that the actual value of the time-stamp is arbitrary, and use an epidemic averaging protocol to give us a system wide arbitrary value which will not change with the addition or loss of a single node. The nodes can then be synchronised using the epidemic averaging function described above.

ESP uses an epidemic protocol to calculate a system wide maximum time-stamp, and to perform the local adjustments. This creates a global synchronised time-stamp like FTSP, but in a fully decentralised way.

The value for the time-stamp is determined by finding and disseminating the largest value time-stamp in the network (see figure 5.5), and then multiplying it by the period every period.

Parameter	All-to-All	Grid
packet size	128 Bytes	128 Bytes
clock period	1sec	1sec
clock drift range	0 - .01	0 - .01
node populations	[4, 8, 12, 16, 20]	[16, 64, 100]
simulation runs	1000	1000
simulation run time	20sec / 96hours	600sec / 96hours
synch window	100ms	100ms
threshold	10ms	10ms
radio noise model	Meyer Heavy	Meyer Heavy

Table 5.1: Parameters for the simulation experiments performed in this chapter, organised by node topology.

This gives us a unified, arbitrary time-stamp, without needing the root node of FTSP.

5.5 Evaluation

To ensure that ESP converges, and that its convergence is stable, I evaluated ESP in simulation and on a testbed. For a synchronisation protocol, time to synchronise is synonymous with time to converge. This also allowed us to evaluate the speed of convergence against other distributed synchronisation approaches like RFA and the amount of communication needed to converge.

The ESP simulation was written in NesC [GLvB+03] and run with the TOSSIM simulator [LLWC03]. I implemented the protocol as an application, and used the standard TinyOS MAC layer. I modified the simulator so that the clock of each node would have a slight skew, or clock drift. I used a time period of one second, which was 1024 clock ticks on the MicaZ oscillator. The drift was simulated by choosing a uniform random value between 0 and .01 which approximates a maximum clock skew of 10 parts per million. This clock skew is typical for the types of oscillators used for timing on the MicaZ [UC10]. The simulations measure the time to synchronise, and the amount of messages needed to synchronise. In the case of the amount of messages, 100% represents each of the nodes sending a sync message every period. Each of the points in the simulation are the averages of 100 simulation runs. The simulations were run on a 64 node cluster computer. In all cases the period of the nodes is one second. The synchronisation window is 100 milliseconds, and the time threshold for message cancellation

is 10 milliseconds. I used two topologies, all-to-all to see how the protocol scales to having a large number of neighbours (deployment density), and a grid topology, with a fixed neighbour population (four in this case) and multi-hop communication. In the all-to-all topology there were populations of four, eight, twelve, sixteen and twenty nodes. I chose twenty nodes as the maximum due to congestion problems that would be caused by high node populations. As stated above, I depend upon the TinyOS MAC layer to deal with medium congestion via CSMA. In the grid topology there were populations of 16, 64, and 100 nodes each one having an $(\sqrt{n} - 2) + \sqrt{n}$ maximum hop depth (where n is the node population).

The first set of results examined the number of synchronisation messages a node needs to hear before it will cancel its synchronisation broadcast. The purpose of this experiment was to assess the degree of synchronisation message redundancy which exists in epidemic synchronisation. A second question is what effect the message redundancy has on synchronisation performance such as time to synchronise. The message thresholds to cancel synchronisation message broadcasts ranged from the reception of four, three, two, and one message.

The results showed that ESP is able to reduce the messages that a node needs to hear in order to synchronise itself to the rest of the network. Reducing the received synchronisation messages for each node either makes no difference to the time to synchronise, or improves the performance by decreasing the time to synchronise. These results are important because they show how much redundancy is involved in regular epidemic protocols, and how that redundancy can be reduced to save communication costs.

In the first graph (figure 5.7) there is no major increase in the time to synchronise for a network with an all to all topology (single hop). The use of random message suppression is faster, but by less than half of a second, and we consider it to be insignificant. All of the results take around the 6.5 seconds. The second graph (figure 5.8) however shows us a clear decrease in the number of messages needed for roughly the same performance observed in figure 5.7. As the neighbour population for each node increases, the number of messages needed to synchronise decreases. Even at the lowest population of four nodes, cancelling after one received synchronisation message halves the percentage of these messages that the network needs to synchronise. With

the largest population of 20 nodes, cancellation after four messages yields more than a 50% reduction in the amount of communication needed to synchronise than if messages had only been cancelled randomly. Cancellation after one message reduced protocol overhead to about 5% of the messages the original RFA protocol would have used.

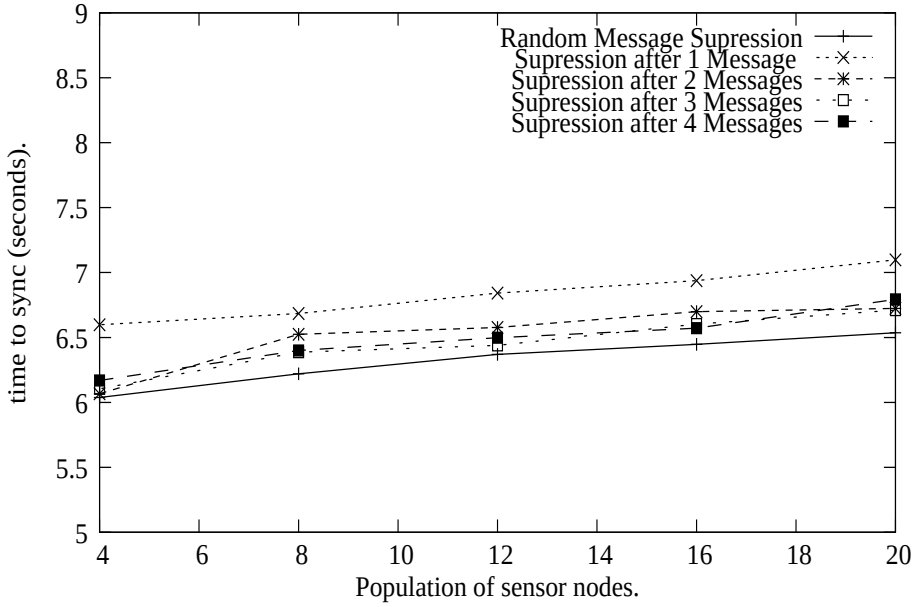

Figure 5.7: Average time for 100% of the nodes to sync in an all to all topology.

The next two graphs, figures 5.9 and 5.10 show the performance of ESP in a square grid topology. In the grid topology, the neighbour population of each node is fixed to a maximum of four neighbours, and the maximum hop count increases from 6 hops for 16 nodes, 14 hops for 64 nodes to 18 hops for 100 nodes. Figure 5.9 shows the time to sync, and as the number of hops increases, cancelling sync messages after hearing one message causes a dramatic increase in the time to sync. With cancelling after two sync messages, the performance is still poorer than with just random message cancellation alone, but only by about 20 seconds. Three and four message thresholds are at worse 10 seconds slower to sync. Figure 5.10 shows us that the reduction in communication is still noticeable, but not as great as in the all to all topology. By the time the communication is reduced to half of what it would be with random message cancellation, the performance degrades considerably.

This result shows us how much communication can be reduced before performance suffers for this particular topology. If the neighbour population of the nodes in a network is high then

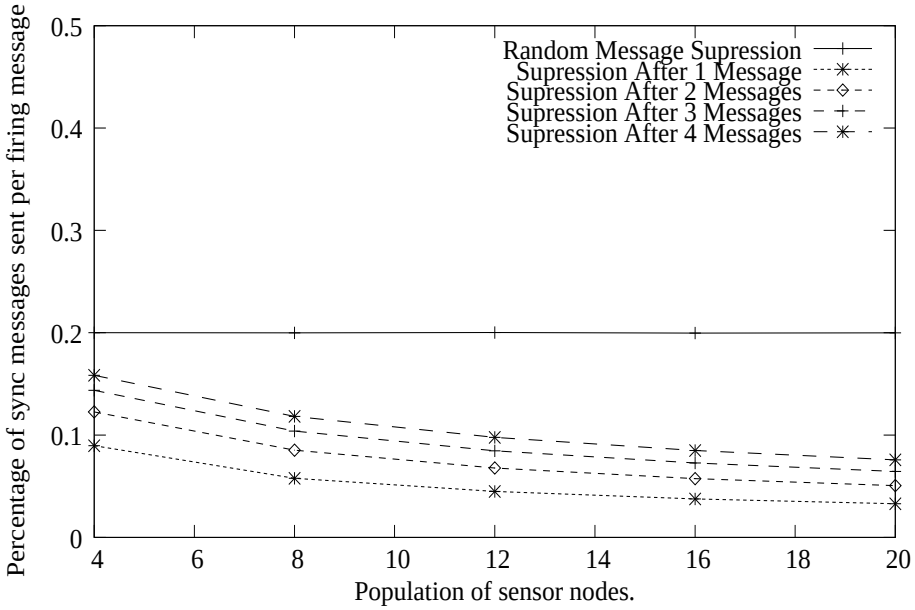

Figure 5.8: Average percentage of firing messages being sent for each firing event in an all to all topology.

the communication overhead, or the number of messages used, can be reduced to as much as 5% of that used by the original RFA. This communication saving comes with minimal effect on the time to synchronise (0.5 seconds for 20 nodes in figure 5.7). When the neighbour population is fixed and the hop count increased in a multi-hop scenario, unsurprisingly, the time to sync increases with the hop count. The messages used, however, remains fixed at that of its neighbour population as shown in figure 5.8. This suggests that communication efficiency will increase with node density and demonstrates in Figures 5.8 and 5.10 that epidemic based synchronisation protocols can be optimised to use less than 80% of the communication in order to function on WSN.

The final simulation based experiment aimed to test the long-term stability of the node's event synchronisation. The simulations were run for populations of four, twelve, and twenty nodes for the all-to-all topology, and populations of sixteen, sixty-four, and one hundred nodes for the grid topology. The nodes synchronised to a one second period. The experiments ran for a period of four days in simulation time. During this time single nodes were randomly removed and added test the robustness of ESP. This experiment also measured the long term affect of clock skew on network synchronisation. Clock skew was set to a random number between 0 and .01 to simulate a maximum clock drift of 10 part per million. I measured the average

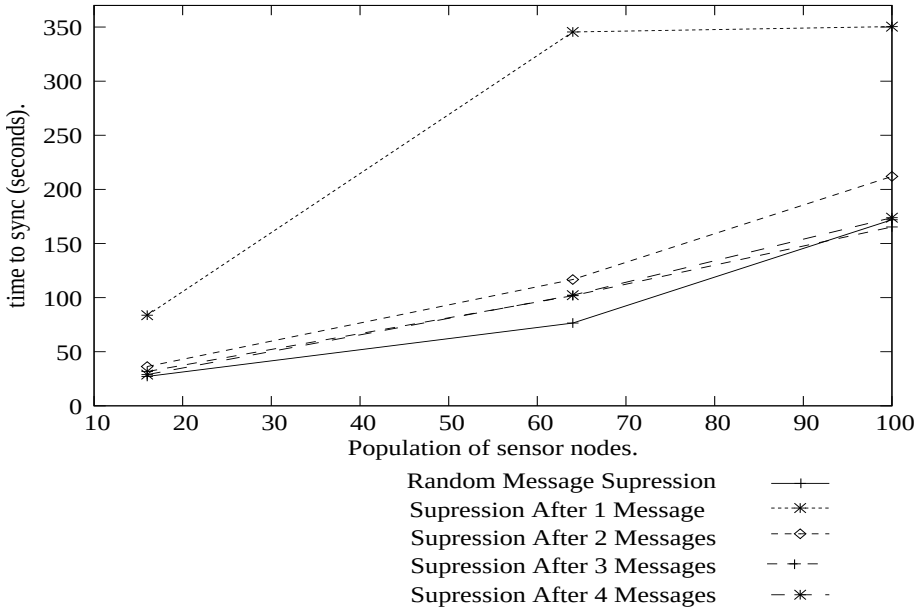

Figure 5.9: Average time for 100% of the nodes to sync in a grid topology.

Topology	4	12	20
All to All	11.6 ms	10.7 ms	12.0 ms

Table 5.2: Long term average synchronisation errors.

synchronisation error over each run, to ascertain if the nodes stayed synchronised over the whole of each experiment. Tables 5.2 and 5.3 show the average synchronisation errors for each experiment.

In no case did the removal or addition of a node destabilise the network and cause long-term instability in the network. If the new node managed to not have its firing suppressed, then that was because it managed to start in sync. These results and the ones presented above demonstrate that the biological computation that I use for ESP converges quickly, and is stable once converged. It is also clear that the message redundancy in epidemic synchronisation can be reduced to enhance performance.

5.6 Test-bed Evaluation.

I implemented the ESP algorithm on a in-lab testbed of MicaZ motes. This was done in order to verify my simulation results, and to further evaluate the synchronisation protocol under the

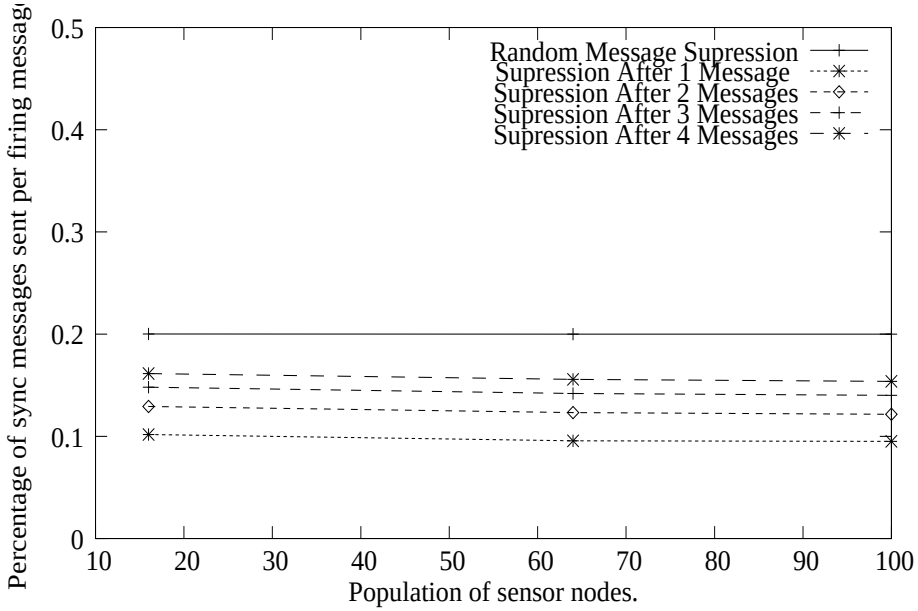

Figure 5.10: Average percentage of firing messages sent for each firing event in a grid topology.

Topology	16	64	100
Grid	14.2 ms	20.3 ms	22.9 ms

Table 5.3: Long term average synchronisation errors.

affects of real radio conditions and with real WSN nodes. I measured the speed of convergence of synchronisation, the time to synchronise and the synchronisation error for different topologies. Time measurement and recording was done using 16 MicaZ motes attached to a logic analyser. When the motes executed an event, they would indicate this via an LED, and this would be recorded by the logic analyser. This gave us an accurate reading of the time the event occurred. The parameters used in these experiments is given in table 5.4.

First I measured the time to synchronise for an all to all topology using no message suppression, and with message suppression. The time was measured in periods to synchronise. Each period was one second long. The results of time to synchronise and the percentage of trials which were successful in synchronising all of the nodes are given in table 5.5.

The results show a performance improvement in time to synchronise when using message suppression. What is interesting is that synchronisation using message suppression is also more reliable than without. This is because a refractory period of half of the total period was used, and without message suppression it is possible for the network to split, with two groups of

Parameter	All-to-All	Grid	Linear
packet size	128 Bytes	128 Bytes	128 Bytes
clock period	1sec	1sec	1sec
TX power	-20dB	-20dB	-20dB
node population	16	16	16
inter-node distance	25cm	25cm	25cm
experiment runs	100	100	

Table 5.4: Parameters for the test-bed experiments which used a logic analyser to measure node synchronisation, organised by node topology.

Suppression	average time to sync	success
None	22.8sec	70%
One Message	4sec	100%

Table 5.5: Advantages of the use of message suppression.

synchronised nodes. Each group out of synchronisation phase with each other.

The next set of experiments looked at the influence of different multi-hop topologies on the time to synchronise and the sync error per-hop. These experiments were performed on the same testbed as the previous experiments with the motes attached to a logic analyser. The topologies tested were all to all with one hop, a grid topology giving a maximum hop count of four hops, and a linear topology with a maximum hop count of 16 hops. The use of a logical analyser meant that the nodes had to be relatively close to each other. There was an inter-distance of 25cm between each node to prevent near-field radio interference using a 2.4Ghz radio, with the power level set to -20dBm. In order to create a multi-hop network I had to specify in the network layer a set of addresses accepted by each node. Even though a node could hear the other nodes in the network, it would only accept packets from, and adapt to, the node specified in its network layer neighbour table.

The results from these experiments are summarised in Table 5.6. They show that there is an increase in synchronisation error for each hop. This error is caused by the latency of the communication stack, and increases with the hop count. In this implementation of ESP the time-stamps were recorded at the application layer, and so were affected by a latency introduced by the radio stack. In later versions of ESP, the time-stamps were recorded at the MAC layer, and the per-hop drift was reduced.

Topology	longest Hop	Avg. sync time	Avg. sync error
All to All	1	4sec	3.2 ms
Grid	4	10.6sec	24.8 ms
Linear	16	20.8sec	45.7 ms

Table 5.6: Summary of sync times for different topologies and hop counts.

The synchronisation stability of the grid topology was similar to that of the all-to-all topology. Once in a state of synchronisation, the network would remain there. Nodes failing or coming into the network also caused no disturbance to the remaining nodes. New nodes would become synchronised themselves with the same average seen in Table 5.6. The synchronisation stability of the linear topology was good as long as there was no disturbance. If a node was restarted, then 50% of the time the node would desynchronise all of its downstream neighbours by partitioning the network. When the network became desynchronised, it would resynchronise itself in the same average time seen in Table 5.6.

To measure the stability of the synchronisation, the network was run for a period of twelve hours. Every node broadcast a message at a random point in their period with the offset of when their period had started, and the number of the message. These broadcasts were received and recorded by a single base-station, which would correlate the start of period times for all of the messages received with the same message number. This way the synchronisation error could be measured every second. During this period all of the nodes remained in a state of synchronisation, with same average synchronisation errors as observed in Table 5.6. The results in Table 5.6 show that the predicted time to converge and stability of the convergence of ESP observed in the simulation results were accurate.

5.7 Global Synchronisation Evaluation

In order to evaluate the accuracy of the time-stamps produced by ESP, I implemented both FTSP and ESP on a testbed of motes. Each mote outputs to a serial port every event period when its event occurs. The output includes the event number, the FTSP timestamp and the ESP timestamp. The difference between the two timestamps is then compared.

Parameter	Indriya	Motelab
packet size	128 Bytes	128 Bytes
clock period	1sec	1sec
TX power	0dB	0dB
node populations	138	89
experimental runs	10	10
experiment run time	4hrs.	1hr.
most neighbours	20	29
max hops	8hops	7hops

Table 5.7: Parameters for the remote testbed experiments performed in this chapter, organised by testbed.

Testbed	Experiment Length	Testbed size	Avg. error	Std. deviation
Motelab	1hr	77	21.16ms	8.57ms
Indriya	4hr	117	49.18ms	198.80ms

Table 5.8: Summary of synchronisation errors observed on two different WSN testbeds.

The experiments were run on two testbeds, the Motelab testbed at Harvard University [WASW05], and the Indriya testbed in Singapore [DCA12]. The experiments were run for one hour on the Motelab testbed, and four hours on the Indriya testbed. The experiment lengths were determined by the quota granted by the testbed administrators.

5.7.1 Global Synchronisation Results.

The results observed from the synchronisation experiments performed on the remote testbeds are shown in table 5.8. Both networks are similar in size, with a maximum network diameter of 7 or 8 hops. The average neighbour degree, or the average number of neighbours for each node is also similar. Indriya has some nodes with 20 neighbours, where the highest neighbour degree in Motelab is 29.

The first point to note from these results is that the global time-stamp drift is similar to that observed from the in-lab testbed where the sensor nodes were monitored with a logic analyser. The average drift was 21.16ms, around 3ms per hop. This time drift represents the drift in time stamps received at the base-station. The significance of this experiment was to show that both event and time-stamp based synchronisation can be performed when synchronisation is viewed as an epidemic biological computation. ESP proved that it was stable, and gave stable

global time-stamp readings for the entire length of each experiment.

5.8 Discussion

I have presented a decentralised synchronisation protocol which works as the result of a biological computation. Epidemic propagation, a form of swarm-intelligence, is the bio-inspired protocol used to create the desired state of synchronisation. The ESP protocol reaches a state of convergence faster than RFA, another example of a bio-inspired decentralised synchronisation protocol. It also has a much lower communication overhead, less than 80% of RFA.

The ESP results suggest that there is communication redundancy inherent in biological computation. Within the world of bio-inspired algorithms, redundancy in general contributes to the robustness observed in real biological systems (like ant and termite colonies). This robustness due to redundancy is well understood with biological systems. There exists the belief that with ubiquitous computing the number of sensors embedded in the environment will greatly increase. There will be a point were there is great redundancy (of some sort) with sensors. At this point efficient, self organising protocols like bio-inspired algorithms will be appropriate. However, given the current resource limitations that we suffer with wireless sensor nodes, such as energy storage and consumption, it is not currently possible to emulate exactly the biological algorithms used in nature, they must be adapted. ESP is an example of such an adaptation.

5.9 Conclusion

WSNs can reveal spatio-temporal phenomenon which have previously been difficult to measure, like the density of smoke in different parts of a bar, or the moisture level in a field at a given point in time. A core requirement of WSN is synchronisation so that every nodes samples can be correlated in time as well as space.

The hypothesis of this work is that wireless sensor nodes can be made to self-organise and

self-manage their resources such as energy or communication bandwidth in an efficient way using biological computation realized through bio-inspired protocols.

I discussed in the background section of this thesis the provision of this and other core services are currently provided by multiple, independent protocols. The previous chapter showed us that the use of multiple protocols can be problematic. If one protocol requires a large amount of communication, it can starve other protocols so that they malfunction. My hypothesis is that to solve this problem services need to be provided as the result of biological computations performed by using bio-inspired swarm intelligence algorithms.

The first piece of evidence in support of my hypothesis was to use already established service protocols, and create a scheduler using biological-style distributed computation where the nodes used their local information and the information of their local, one-hop neighbours to schedule which protocol to send, and when to access the communication medium.

The greedy scheduling policies could be effective, but would always be constrained by the capacity region of the communication network. In large, multi-hop networks with heavy communication traffic, the schedulers would cause message delays and as a result FTSP would fail. My next piece of evidence in support of a bio-inspired approach was to try and combine, and therefore reduce, the communication of service protocols. I did this by basing the service protocols themselves on bio-inspired algorithms.

The first service protocol is called ESP and provides time synchronisation based on epidemic information propagation. I demonstrated how using message based broadcast suppression could reduce the communication overhead of synchronisation by 80% compared to other similar approaches such as RFA. Furthermore, ESP is also capable of performing global synchronisation, where all such similar attempts only synchronise events. This is done by computing a global time-stamp as the results of a distributed computation using an epidemic algorithm. This was my first example of how the result of a biological computation can be used to provide a service to a user application and synchronise the nodes in the network.

In the next section I provide the final piece of evidence in support of my hypothesis and re-

turn to the provisioning of multiple services through the use of biological computation. I will add another service, information dissemination. The dissemination service will be based on the same epidemic propagation algorithm as my synchronisation protocol. More importantly, dissemination will work along side of my synchronisation primitive without incurring any additional communication cost. I also explore how reducing the number of control messages in the network can both speed up the convergence of the biological computation, and reduce the messaging overhead of the WSN management layer.

Chapter 6

Providing Multiple Services With Intelligent Combination

6.1 Introduction

The illustrative smoke monitoring application will require more services than just synchronisation and data collection. Suppose that an application requires that the rate of sampling changes as a function of the population of the bar. As more customers enter, more smoke will be produced, and there will be more moving bodies to churn up the air. The application needs to increase the rate at which the nodes sample the air quality in order to adapt to the change in environment. A new sampling frequency must be disseminated throughout the network, the nodes need to re-synchronise themselves, and then return to work as soon as possible.

In this section I provide the final bit of evidence to support my hypothesis that low-resource devices such as WSN nodes can be made to self-organise and self-manage in an efficient and robust way through seeing management as a biological computation which is realised through the use of bio-inspired algorithms. To show this, I add dissemination to the WSN system layer by building on top of ESP using a form of protocol combination that I refer to as intelligent combination. I demonstrated in chapter 4 that Unified Broadcast was very successful at reducing the total amount of communication used in a network. This represented a non-scheduling

approach to keeping communication within the capacity region. The problem with UB was that it combined broadcast communication without any consideration for the functioning of the protocol. Bio-inspired algorithms provide a more intelligent way to combine protocol traffic. I combine the protocols themselves, so that neither experience any disturbance. In doing so I return to the earlier problem of the combination of multiple protocols and their tendency to interfere with one another. I demonstrate how bio-inspired swarm intelligence algorithms can have their communication combined since they use information in the same way.

To help prove my hypothesis I continue my investigation into the WSN management problem, that is the extent to which I can reduce the communication of the management layer while offering more services, and the effects that that will have on performance. I refer to this extension as FiGo, and use it to demonstrate how the use of bio-inspired algorithms can aid the combination of functionality provided in a network with out incurring a large increase in communication overhead.

I refer to my WSN operating system as FiGo (for Firefly and Gossip). FiGo arose from attempts to fix problems which were encountered in actual deployments, as well as having its roots in experimental algorithms. At the beginning of chapter 4 I presented experiments showing the failure of more than one protocol working together at the same time. The motivation for these experiments arose during an actual WSN deployment. The FiGo operating system, which I present in its current form here, was the solution which I developed to solve this problem, and provide multiple services to a WSN application.

6.2 State of the Art

The Trickle algorithm [L+03], [LBC+08] was discussed in chapter 5, and will be discussed again here in greater detail. Trickle forms the core of many well used protocols such as: Deluge [HC04b] for code propagation; Drip, which is used by Deluge and by the Sensor Network Management Protocol (Nucleus) [TC] to propagate commands; Tenent [GJP+06] to distribute scripts; and the RPL IPv6 routing protocol for wireless sensor networks [Win12].

Drip uses a Trickle timer [L+03] and is a gossip based algorithm. Each node has a piece of code, with a version number. Time is broken up into periods with a duration of τ. At a point within that period, randomly chosen between the times $[\tau/2, \tau]$, a node will broadcast the version number of the code it has. Trickle uses 'polite gossip' meaning that if a node hears an announcement for the code version that it has before it reaches its random broadcast point, it suppresses its announcement. If a node hears an announcement for a code version newer than the one it has, then it broadcasts its older version number to trigger an update. The first node to hear the older code version number (any node who has the most recent code version, not just the one which broadcast the most recent version number) will broadcast a code update.

There is a relationship between the length of the period τ, the speed of code propagation, and the protocol overhead incurred. The shorter τ, the faster new code is propagated, but the higher the overheads (announcement overheads are sent more frequently, so more packets will be sent over time). Conversely, the longer τ, the slower updates propagate, and the lower the overhead. Trickle uses this fact by utilising positive and negative feedback to dynamically change the value of τ. If a period τ goes by, and a node has not needed to broadcast because there is no new data in the network, then it doubles its τ. It continues to double its τ every period that goes by until it reaches a maximum value τ_{max}. When a node hears a broadcast with a newer code version, then it halves its τ. This continues until the value reaches a lower limit of τ_{min}. This way the value of τ dynamically adapts itself to propagate code quickly when there is an update to propagate, and slows down when there is nothing new.

Drip provides channels that application components can register to, and listen on. The registered component provides the message buffer, and data received on a channel is delivered directly to the component buffer. Drip provides both named and unnamed reliable message dissemination. The Drip message contains three extra fields: node name, group name, and time to live. A message with these three extra fields is passed to the naming component, and if it pertains to the receiving node it is acted upon.

A similar gossip based algorithm was used in the dissemination protocol of the Impala WSN management system. Impala was used in the ZebraNet Project [LSZM04], monitoring zebras

in Kenya. This system used absolute clock synchronisation which it obtained by equipping each sensor node with a GPS module, and synchronising the clocks to the GPS satellites. FiGo wraps its dissemination protocol in an event synchronisation mechanism, thereby reducing its communication cost to the system.

A protocol to maintain state consistency in distributed systems called GoSyP is proposed in [RKRK]. It is based on a unicast epidemic protocol, and compares itself to polite gossip. The description of polite gossip mentioned in the paper did not mention message suppression, the core mechanism which makes polite gossip 'polite'. The related work section mentioned the Trickle protocol and commented that it used a process which reduced total data sent, but did not specify how it did that. The paper failed to mention that Trickle uses polite gossip with a variable communication window. The evaluation shows that GoSyP has a lower dissemination overhead than broadcast epidemic propagation exemplified by polite gossip. Evaluation was only performed in simulation, and for radios using 802.11b with a MAC layer using RTS/CTS to mitigate hidden terminal problems. I do not feel that this is an adequate evaluation to properly ascertain the communication overhead of GoSyP.

An interesting protocol combining information dissemination and synchronisation is presented as the GLOSSY protocol [FZTS11]. It uses constructive interference by having all of the nodes transmit their dissemination packets at the same time. Constructive interference means that as long as all nodes transmit within 5 microseconds of each other, then collisions will not affect the ability of receiving nodes to correctly decode the message. The evaluation of this protocol shows very fast dissemination times and very low synchronisation error.

The problem with GLOSSY is that it requires very reliable synchronisation in order to function properly. Analytical results are presented to suggest that the necessary synchronisation can be achieved, and testbed results confirm this in a laboratory environment. Whether these results can be maintained in a real WSN deployment on low-cost sensor nodes remains to be seen. Another question is the ability of GLOSSY to share the network with other protocols. The authors clearly state that GLOSSY needs complete control of the network when it is functioning. As I show earlier in this work, this assumption can be dangerous. The final issue with GLOSSY

is that it requires that all the nodes communicate. In my view this is unnecessarily redundant in terms of communication required.

6.3 Information Dissemination with FiGo

The FiGo WSN operating system uses biological computation to manage a WSN. The results of the computation is that all of the nodes of the WSN are in a current, consistent state. Current with respect to the latest command, and consistent by having the current command be present on every node in the network. The nodes achieve this by using epidemic propagation and local node processing of local neighbour data with no central control.

FiGo offers a synchronisation service and a command dissemination service with a low communication overhead. The protocol works by combining synchronisation and dissemination information into the same packet. This combination works because the synchronisation and information dissemination protocols both use information in the same way as described in chapter 3.

The FiGo algorithm is shown in pseudo-code in figure 6.2. Every period, every node sets a random timer to fire between now and the end of the synchronisation period. When the timer fires, the node sends a synchronisation message. Included in the synchronisation message is the metadata which describes the version of the information which the node currently has. This information refers to the command information which a user or application would like to keep unified across the entire network. When a node receives a neighbour synchronisation method it parses it as two pieces of information. The node uses the synchronisation information to synchronise to, and the dissemination information to determine its appropriate action.

The synchronisation information is treated in the same way as in chapter 5. When a synchronisation message is received during the refractory period, the synchronisation information is ignored. The synchronisation information is only used when it is received after the end of the refractory period. In this case, the refractory period is the first half of the synchronisation

period. So, synchronisation information is only used when the message is receive in the second half of the period.

The dissemination data is parsed and used during the entire length of the synchronisation period. If the data received is the same as the local data, then no further action is taken with regards to dissemination information this period. If the information received is out-if-date, then the local node sends the newer data. In this way the node which previously had out-of-date data would now be up to date, and it would increment its local metadata to reflect this. If the received metadata is newer than the local metadata, then the local node sends its metadata immediately, triggering the update mechanism mentioned above. If a node overhears old metadata it uses a short random timer before it responds. If it hears a suitable response before its timer fires, then it cancels its own response, assuming that another node has updated the out-of-date node.

The use of timers and overhearing allows the same form of message suppression as was seen in the synchronisation chapter. This time, a node's sync/metadata message is cancelled if it hears a sync message before its timer has fired and the value of the metadata on that sync message was the same as its own. The difference between the ways the data are handled during the refractory period and after the refractory period are illustrated in figure 6.1.

In the case of simple data like frequency length, the data, along with some associated metadata (like the age of the data), can be parsed and used directly. If a node receives a message with metadata greater than its local metadata, then it changes its frequency length with that of the message. In the case of the dissemination of larger data, the extra information included in the synchronisation message can be just metadata which references other data that then needs to be communicated separately. The data used by FiGo (the event frequency) is small enough to be included in the sync message along with the metadata. However, my evaluation version is implemented such that the new data needs to be requested and downloaded separately. This was the more challenging way to implement FiGo, and gave us 'worse case' performance figures because of the extra overhead needed for the separate data-transfer protocol to work.

The code savings made by my protocol is made by the obvious fact that the combining all of

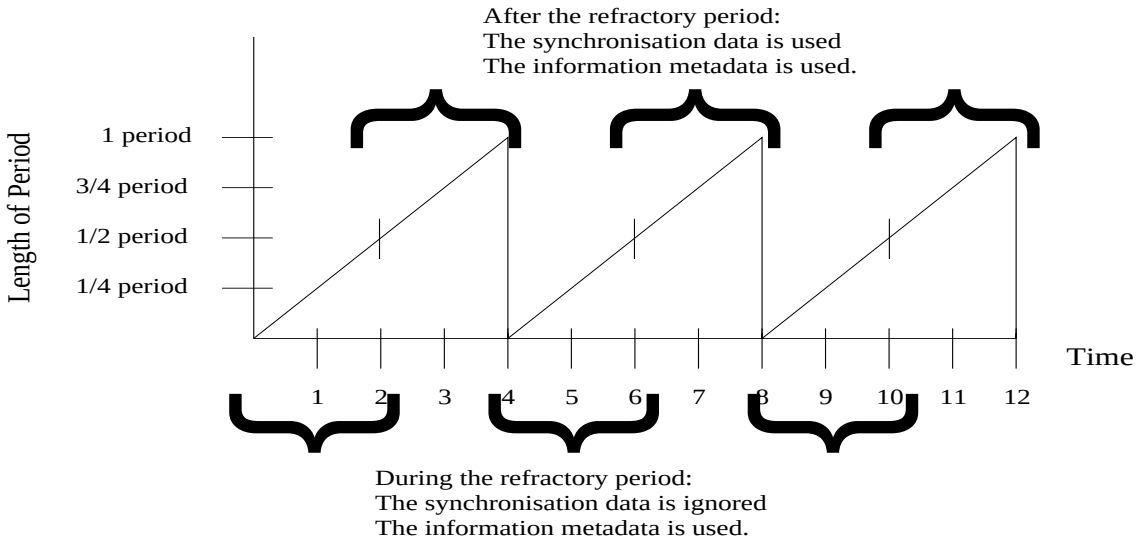

Figure 6.1: Diagram explaining the way data from a message is used if it is received during or after the refractory period. A message received during the refractory period has its sync information disregarded, and its metadata information regarded. After the refractory period, all information is regarded.

the data onto one message results in less transmissions than sending multiple protocols. This is clearly stated in the notation from the model provided in section 4. I use $r_x^p(t)$ to denote the rate at which protocol p produces new messages on node x during time period t. Simply stated, the summation of all messages produced during a time period t is greater than combining all of the data into a single, combined message.

$$r_x^1(t) + r_x^2(t) + r_x^3(t) > r_x^{(1+2+3)}(t)$$

Related to this is the fact that the size of each protocol's queue will be combined into one queue, and will save memory on the nodes.

$$Q_x^1(t+1) + Q_x^2(t+1) + Q_x^3(t+1) > Q_x^{(1+2+3)}(t+1)$$

I evaluate the ability of this protocol to converge, and the stability of that convergence through experimentation, both in simulation and on real sensor nodes.

It is important to remember at this point that I am not just combining information here,

```
local_clock = 0
cycle_length = 100
refractory_period = cycle_length/2
duty_cycle = cycle_length
next_broadcast = random(0, cycle_length)
local_metadata = 0
same_count = 0
loop
  if local_clock == next_broadcast and same_count < same_threshold  then
    transmit local_clock and local_metadata value
    restart at top of loop
  end if
  if clock ≤ cycle_length then
    listen
    if A message is overheard AND local_clock > refractory_period then
      adjust local_clock to average of local_clock and time in message
    end if
    if The message contains metadata > local_metadata then
      transmit metadata now
    else if the message contains metadata < local_metadata AND same_count < 1 then
      transmit data now
    else if the message contains metadata == local_metadata AND time in message ==
    local_clock then
      same_count = same_count + 1
    end if
  end if
  local_clock = local_clock + 1
end loop
```

Figure 6.2: Pseudo code algorithm for the FiGo algorithm

but I am doing so while maintaining protocol performance. This is possible because both protocols treat information in the same way. Both protocols use local only information, and both do processing on the local node. This allows both protocols to share communication at no performance penalty to either, and is what I refer to when I use the term "intelligent combination".

6.4 FiGo Evaluation.

The work in the previous section was all about reducing the communication costs of bio-inspired algorithms for synchronisation. The results in this section are about the performance effects

Parameter	All-to-All	Grid
packet size	128 Bytes	128 Bytes
clock period	1sec	1sec
clock drift range	0 - .01	0 - .01
node populations	[4, 8, 12, 16, 20]	[9, 16, 25, 36, 49, 64, 81, 100]
simulation runs	1000	1000
simulation run time	60min	60 min
synch window	100ms	100ms
periods used	[500ms, 1000ms, 2000ms]	[500ms, 1000ms, 2000ms]
radio noise model	Meyer Heavy	Meyer Heavy

Table 6.1: Parameters for the simulation experiments performed in this chapter, organised by node topology.

of using bio-inspired algorithms to offer multiple services while keeping a low communication overhead. FiGo intelligently combines event synchronisation and information dissemination by noting that both algorithms treat information in the same way by using similar bio-inspired protocols to produce their computational result. This similarity means that the communication can be shared without negatively affecting either protocols performance. With the synchronisation part, this means not affecting time to synchronisation. For the dissemination part this means not affecting time to disseminate.

The evaluation of FiGo is concerned with two things. The first is the ability to disseminate data and synchronise the network. This is a functional requirement of my distributed WSN operating system. The second thing is the ability to perform this function cheaply in terms of communication costs measured by the number of messages used. I also look at how the performance or time to converge of the management layer is affected by the requirement to reduce communication costs.

FiGo was evaluated both in simulation and on real WSN nodes. The simulation experiments will be presented and discussed first. Following the simulation results I will present the results that I obtained through experimentation with FiGo on real WSN nodes.

6.4.1 FiGo Simulation Evaluation.

In this section I present my simulation results evaluating the performance of FiGo. The simulation was written in NesC [GLvB+03] and run with the TOSSIM simulator [LLWC03]. I use the standard TinyOS Active Message interface and MAC layer. The simulations output two metrics: The time to disseminate a new event period and synchronise to it for all of the nodes, and the percentage of messages used to disseminate and synchronise. The new data which is disseminated is new event periods.

The simulations were performed on a 64 node cluster computer. Each result of the simulation was the average of 1000 simulation runs. An individual simulation was run for 60 minutes of simulation time. In the simulation, all nodes started with the same firing period(frequency), but were all out of phase with each other. First I measured their time to synchronise. Once the nodes were synchronised, I waited for three minutes, and then disseminated a new firing period. I then measured the time from the beginning of the dissemination to the point at which all of the nodes were synchronised with the same period and phase. Three event periods were used: one second(1000ms), two seconds(2000ms), and half a second(500ms). The results shown were the times to synchronise from random start, and from each of the period changes.

I used two topologies, 'all-to-all' to see how the protocol scaled to having a large number of neighbours (deployment density), and grid topology. The grid topology had a fixed neighbour population (four in this case) for the inner nodes, three neighbours for the nodes on the edges, and two neighbours for the nodes on the corners. Grid topology also allowed us to test multi-hop communication. In the all to all topology I used populations of 4, 8, 12, 16 and 20 nodes. In the grid topology I used populations of 9, 16, 25, 36, 49, 64, 81, and 100 nodes each one having an $(\sqrt{n} - 2) + \sqrt{n}$ maximum hop count (where n is the node population). The hop count for each node population is shown in table 6.2.

I varied the message suppression in three ways to see its impact on the time to disseminate and synchronise after the frequency changes, and percentage of messages used per firing event. The first way the messages were suppressed was randomly with a probability of 20%. This was

Nodes	Hops
9	4
16	6
25	8
36	10
49	12
64	14
81	16
100	18

Table 6.2: Number of hops per nodes in the grid topology.

to compare against a purely random policy. The next two suppression methods suppressed a node from sending if it received messages with the same time and data payload. The number of messages needed to suppress communication were two messages and one message.

6.4.2 Single-hop Simulation Results

I observe in Figures 6.3, 6.4, and 6.5 that FiGo with random message suppression and fixed message suppression have very similar dissemination and synchronisation times. With a population of 20 nodes and an event frequency of 1000ms FiGo with fixed suppression was a least a half a second faster. The event frequency of 2000ms showed that FiGo with fixed message suppression was up to 3 seconds faster than FiGo using random message supression. With the frequencies of 1000ms and 2000ms, FiGo with a message threshold of one had a lower synchronisation time than a message threshold of two. These results show that there is very little difference between the synchronisation times for FiGo using random message suppression or fixed message suppression. The real difference can be seen in communication overhead, here shown as the percentage of sync messages used per firing event.

Figure 6.6 shows results similar to the comparison of random message suppression to fixed message suppression in the synchronisation chapter 5, Figure 5.8. The message overhead required by FiGo with fixed message suppression is at worse the same as random message suppression for a population of 4 with a two message suppression threshold, and slightly better with a message suppression threshold of 1. The messages used by FiGo with fixed message suppression steadily decrease as the neighbour population increases. By the maximum node density of 20

nodes, a message threshold of 1 is using half of the messages as FiGo with random message suppression. This shows that very low amounts of communication are needed to successfully disseminate and synchronise to new event frequencies.

If the time to sync is calculated in cycles (time to sync divided by cycle time in seconds), fixed message suppression with a threshold of one message requires about half of the number of cycles of random message suppression at a period of 2000 milliseconds in Figure 6.4. In Figure 6.5 the time to sync is actually faster than the one with synchronisation on its own, but the number of cycles used is about the same.

The increased speed of synchronisation observed in Figure 6.5 is not surprising considering that the nodes were synchronised when the new frequency was introduced and disseminated. Another thing that this result confirms for us is the stability of the synchronisation algorithm to disturbances. When the event period is changed the network is forced out of synchronisation due to the fact that not all of the nodes have the same event periods. However, the network quickly converges again to a state of synchronisation once all of the nodes have received the new event period. I attribute a large component of the time to sync seen in both Figures 6.5 and 6.4 to the dissemination of the new frequency.

These results show us that I can intelligently combine frequency dissemination with phase synchronisation. The addition of dissemination does not affect the time to synchronise or the stability of the synchronised state, nor does it increase the communication overhead in a single hop network. The results also show us that communication overhead can remain fixed or decrease as neighbour population increases. My next set of results look at the effect of multi-hop communication on time to disseminate and synchronise, and well as the message overhead.

6.4.3 Multi-hop Simulation Results

Figures 6.7, 6.8, and 6.9 give the time to disseminate and synchronise a multi-hop network with a grid topology for each of the three sync periods in order. They compare the sync times of

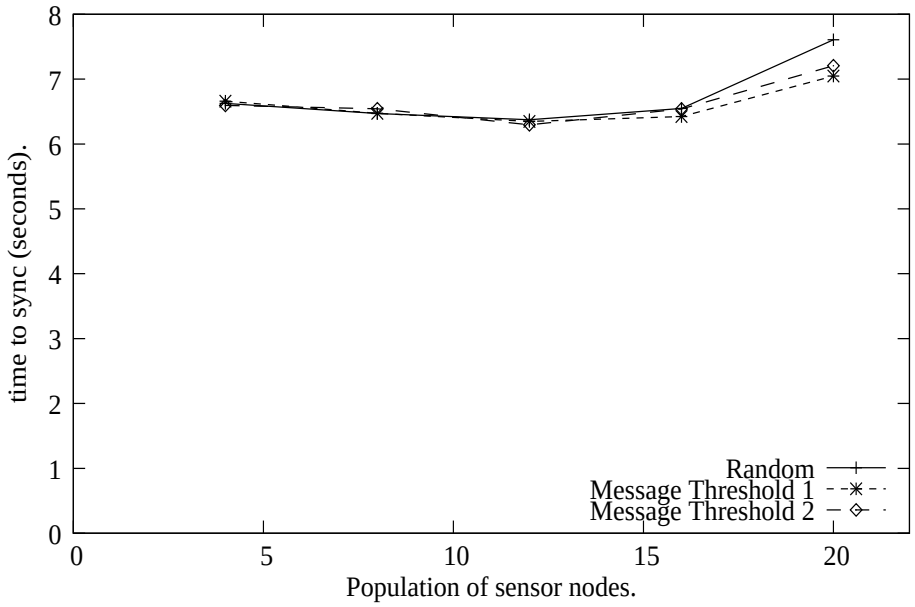

Figure 6.3: Comparison of dissemination and sync time of FiGo with random suppression and message suppression with two different message cancelling thresholds. The results represent the time to disseminate a new frequency and sync to it. The period length is 1000 milliseconds. Average time for 100% of the nodes to sync in an all to all topology.

FiGo using random message suppression against fixed message suppression of thresholds of 2 and 1 message. In all cases the times to sync are almost identical. Both fixed thresholds take marginally longer to sync in all cases, yet there is a marginal reduction in messages broadcast during the same periods.

Another point worth discussing is that the number of cycles needed to synchronise from a random start in Figures 6.7, 6.8 and 6.9 are nearly identical. By dividing the time to sync by the period (in seconds), Figures 6.7 and 6.8 are the same, and Figure 6.9 takes slightly less cycles. This is not unusual, because before the new period is introduced into the network, the nodes are synchronised. The rate of dissemination is therefore very quick, and the nodes are able to re-sync themselves rapidly. Therefore, I attribute much of the time to dissemination time. This result also shows the stability of the synchronisation part of the protocol with respect to disturbances such as, part of the network changing its synchronisation period.

Figure 6.10 gives the average number of messages used for each time period. From the point of view of messages used, all node populations are the same. Once again I look at the percentage of messages sent per firing event. If all of the nodes sent synchronisation messages the messages

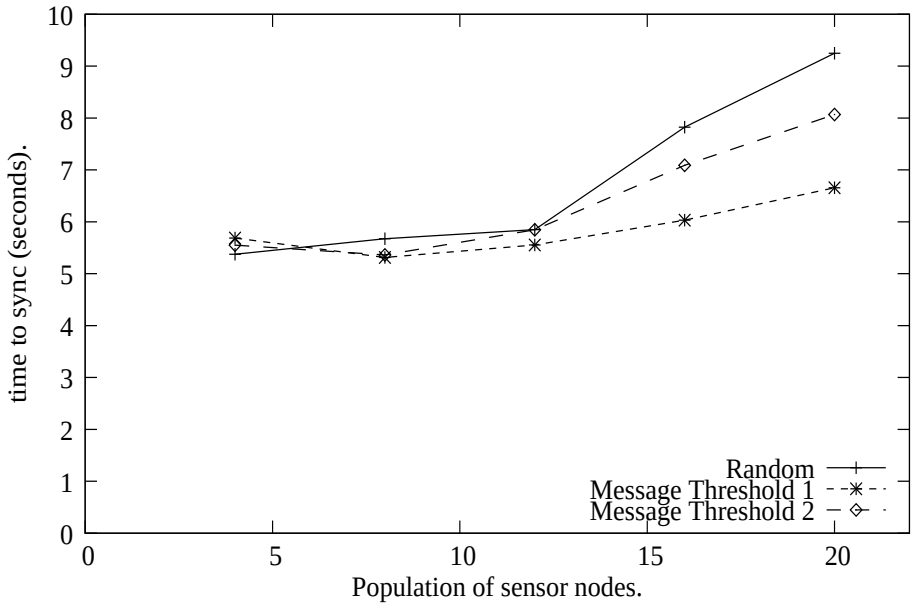

Figure 6.4: Comparison of dissemination and sync time of FiGo with random suppression and message suppression with two different message cancelling thresholds. The time represents the time to disseminate a new frequency and sync to it. The period length is 2000 milliseconds. Average time for 100% of the nodes to sync in an all to all topology.

used would be 100%. Fixed message suppression with a threshold of two gives slightly better performance than random suppression. When the message threshold is reduced to one message, fixed suppression gives better performance that random suppression. This is offset by a slightly longer time to disseminate and synchronise as seen in figures 6.7, 6.8 and 6.9. An interesting result is that the population of the nodes did not affect the percentage of control messages used in the network. This is the same result that I demonstrate in the single-hop results.

All of the simulation results show that my bio-inspired swarm intelligence protocol lends itself well to the combination of different types of data without impeding the functioning of any individual protocol. The bio-inspired approach is capable of offering different services with only a modest communication overhead due to the way that bio-inspired algorithms use information. Furthermore, the communication overhead appears to increase linearly with network population.

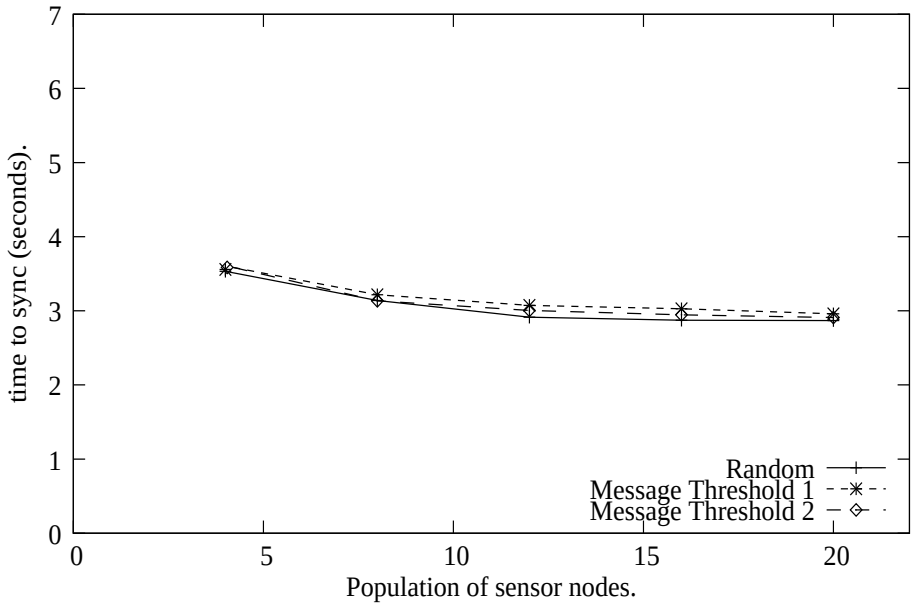

Figure 6.5: Comparison of dissemination and sync time of FiGo with random suppression and message suppression with two different message cancelling thresholds. The time represents the time to disseminate a new frequency and sync to it. The period length is 500 milliseconds. Average time for 100% of the nodes to sync in an all to all topology.

6.5 Implementation of FiGo on Motes.

Evaluation of the FiGo protocol was also done on real sensor node hardware. My first experiment was to confirm my simulation results for time to synchronise. My second set of experiments implemented FiGo on a remote WSN testbed of 117 Telosb motes to test the amount of control messages used, and the time to disseminate new information on a large scale multi-hop network.

6.5.1 Confirmation of Synchronisation.

In order to verify my time to synchronise simulation results, I implemented FiGo on 9 MicaZ motes, and arranged them in a 3 by 3 grid. I used the same code as the simulations, and emulated a multi-hop network by specifying in the network layer the neighbours from which each node would accept communication. The approach gave us a grid topology, where each corner node had two neighbours, the edge nodes had three neighbours, and the centre node had four neighbours. In order to instrument the network and measure time to synchronisation,

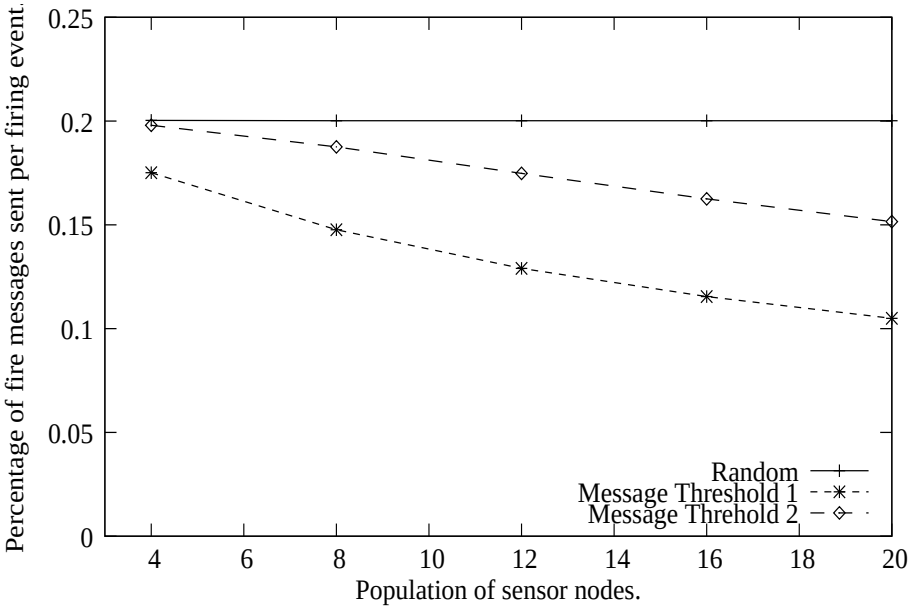

Figure 6.6: Comparison of message usage of FiGo with random suppression and message suppression with two different message cancelling thresholds. These results are the same for all period lengths. Average percentage of firing messages being sent for each firing event in an all to all topology.

the LEDs were flashed to indicate the occurrence of an event. The network of nine motes was then filmed with a Photron Fastcam SA1 [Pho09] recording at 500 frames per second. I was able to determine with an accuracy of 2 milliseconds when the nodes reached synchronisation. A video camera was used to instrument the nodes so that I could observe the nodes with out affecting their operation.

The network of 9 nodes in the grid topology converged to synchronicity in 6 seconds, similar to the time predicted by the simulation. This is interesting because the simulation used the Meyer Heavy noise trace to model radio interference. This was created by sampling the received signal strength register(RSSI) of a CC2420 radio chip on a MicaZ mote at 1kHz in the Meyer Library at Stanford University [LCL07]. The heavy noise trace was created by taking samples while a large load was artificially created on the libraries WiFi network, and represents an extremely noisy radio environment. The radio interference of the testbed was much less because the nodes were in the front of the Blackett Physics laboratory which had only one possible source of interference (a WiFi hub) and there were no users apart from myself and the camera operator present during the duration of the experiments. All of the nodes had a strong signal due to their proximity. This suggests that noisy environment created by using the Meyer Heavy noise

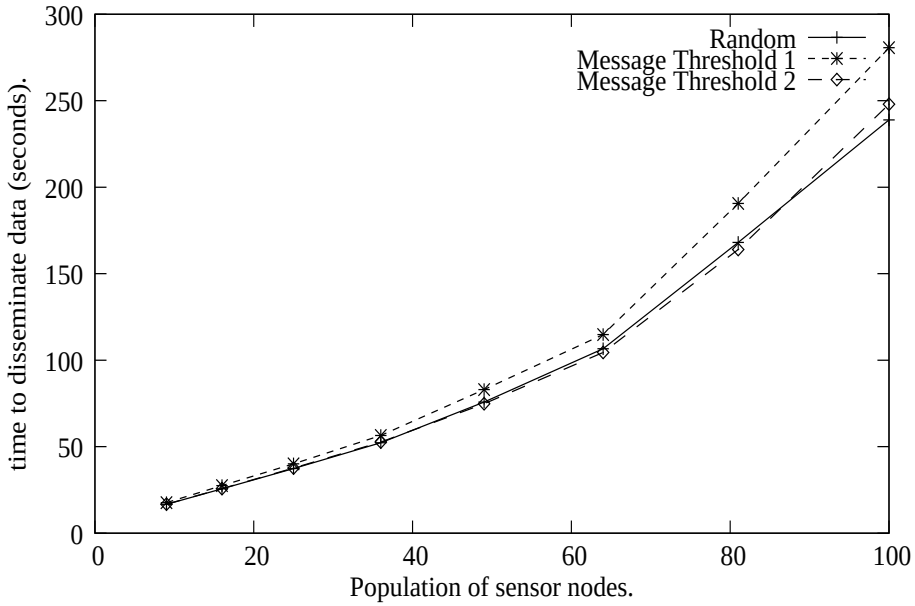

Figure 6.7: Comparison of the time to disseminate new data with an event period of one second. FiGo with both random suppression and message suppression at different message cancelling thresholds is shown.

trace to model the radio environment did not reduce the performance of my synchronisation protocol.

Figure 6.11 is a still of the video frame where the nodes flash their LEDs (lower left-hand corner of sensor board) within the synchronisation tolerance. Please note that the time-stamp (last value in the right-hand column at the top of the image) shows all sensors on at 00.00.21.668 (twenty one seconds and six hundred and sixty eight milliseconds, with a frame taken every two milliseconds). This is because the last node is turned on at 00.00.15.760 milliseconds (the moment my hand flicked the last sensor node switch), and I use that time as the start point.

This experiment confirms that FiGo still functions on real hardware, even after the inclusion of the dissemination service. This experiment was performed as a sanity check to ensure that the extension of my synchronisation protocol from the previous chapter still performed as expected on real hardware. The next set of experiments moves the same protocol and implementation onto a remote WSN testbed to evaluate both synchronisation and dissemination performance on a larger network using a real application.

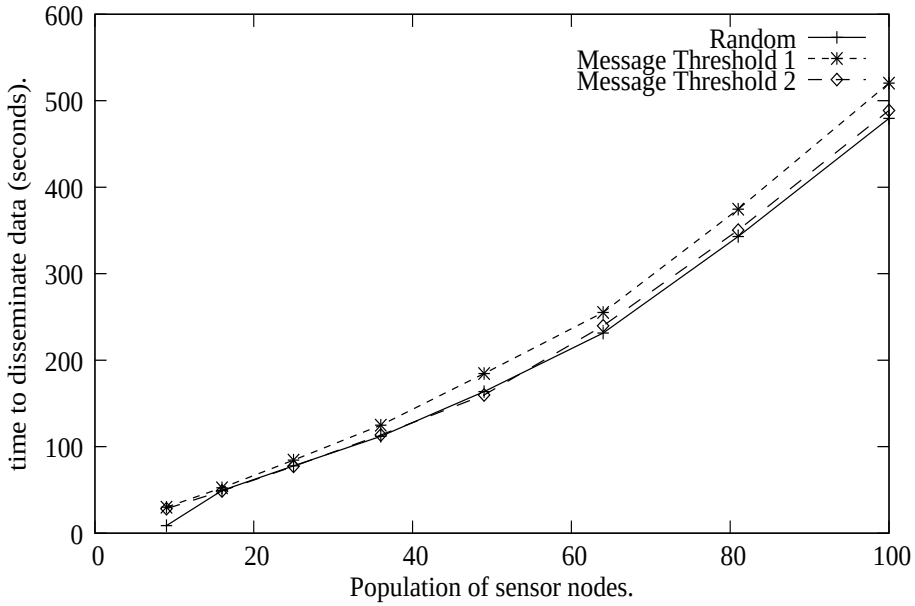

Figure 6.8: Comparison of the time to disseminate new data with an event period of two seconds. FiGo with both random suppression and message suppression at different message cancelling thresholds is shown.

Parameter	Indriya
packet size	128 Bytes
clock period	1sec
TX power	0dB
node populations	117
experimental runs	10
experiment run time	4hrs
most neighbours	20
max hops	8hops

Table 6.3: Parameters for the remote testbed experiments performed in this chapter.

6.5.2 FiGo Experiments on a Remote Testbed.

The next set of experiments were run on the Indriya [DCA12] wireless sensor network testbed. All 117 Telosb nodes (at that time Indriya had 117 functioning nodes) were programmed with the same code which consisted of a temperature sensing application which took synchronised samples every second and sent them to a single base-station every second. Every thirty seconds, a dissemination root would disseminate a command to change the colour of the LED. The LED was flashed every second to indicate that a temperature sample had been taken. I had two versions of the application. One used the Drip dissemination protocol and the FTSP time synchronisation protocol from the TinyOS library and represented the approach and perfor-

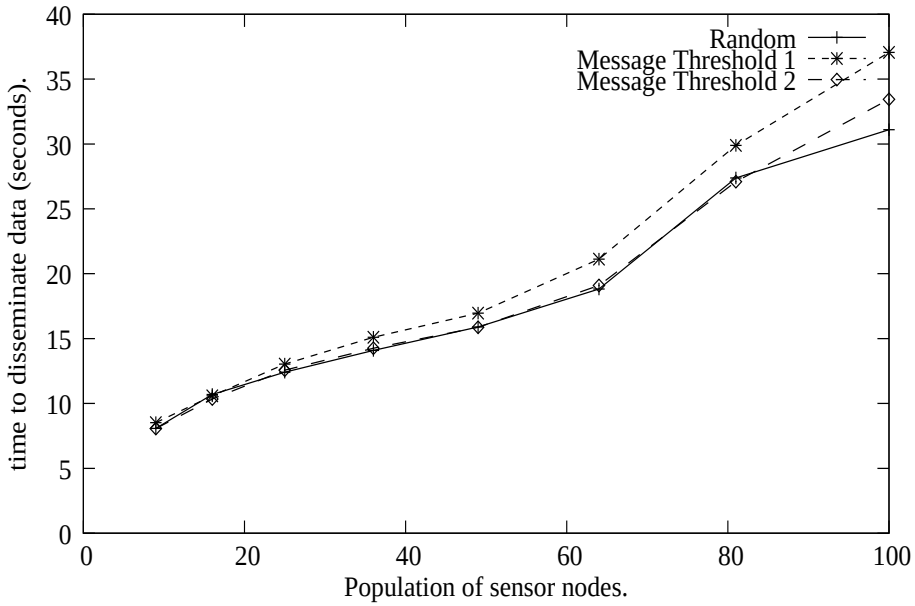

Figure 6.9: Comparison of the time to disseminate new data with an event period of half a second. FiGo with both random suppression and message suppression at different message cancelling thresholds is shown.

mance of WSN management systems like Nucleus. The other version used FiGo. I compared the number of control messages used by each application, and the time to disseminate new information.

The temperature sensing application was run for four hours, five times. Both applications used CTP for data collection. FiGo sent a synchronisation pulse every second, and Drip used a TrickleTimer with a maximum period of one second. FTSP sent a sync pulse once every three seconds. These sending rates show the performance of FiGo under high load conditions.

6.5.3 FiGo Testbed Results.

These experiments compare FiGo against combinations of the service protocols commonly used by the WSN management protocols, such as Nucleus, discussed in the background section 2. By evaluating against these combinations I also compare the performance of various aspects of FiGo against the performance of the WSN management protocols mentioned in the literature.

My first set of testbed experiments were a comparison of the time in seconds for each protocol to disseminate a new LED colour. In this case I compared Drip against FiGo. The results

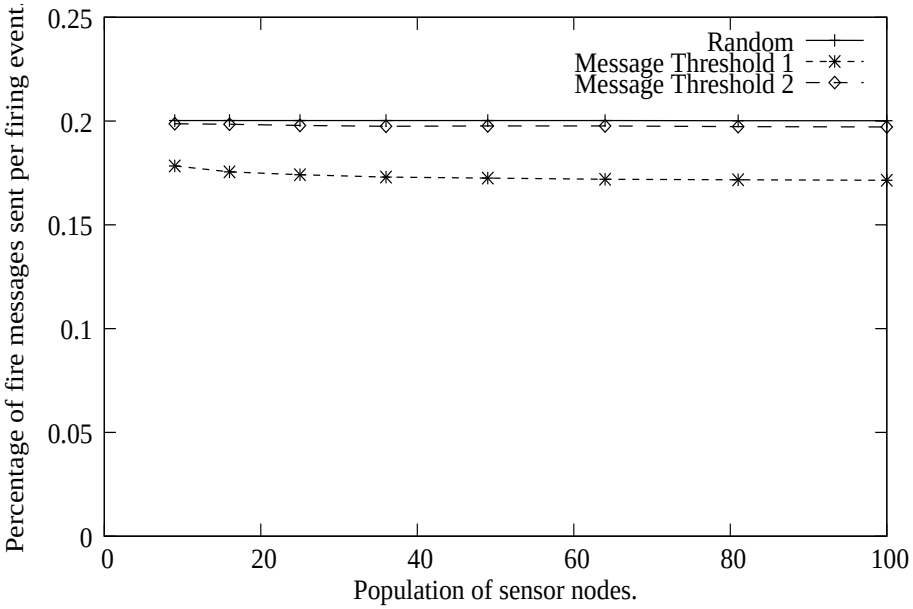

Figure 6.10: Comparison of messages used by FiGo with random suppression and message suppression. Shown are the messages used from a random start to the introduction of a new event period of two seconds. The nodes are organised in a grid topology.

Protocol	Avg. Time	Std. Deviation	Avg. Messages Used
FiGo	25.79sec	8.37sec	416,668
Drip	21.07sec	9.24sec	805,968

Table 6.4: A comparison of dissemination times in seconds on the Indriya Testbed.

are presented in table 6.4. These results show us that on average Drip disseminates code 18% faster than FiGo. This result is not surprising because Drip uses a Trickle timer. A Trickle timer sends data advertisements at a fixed rate when there is no new data in the network. When a node detects (or receives) new data, then the Trickle timer reduces the rate at which advertisements are sent, thereby increasing the speed at which nodes discover that they need new data. FiGo always maintains the same advertisement rate because it is coupled to the synchronisation mechanism.

The increased advertisement rate of Drip makes it disseminate new data 18% faster. This increase in speed does come at a cost. In the final column of Table 6.4 I show that the increased speed of dissemination required 48% more messages. Added to this is the fact that messages FiGo is using are also providing synchronisation information at the same time.

My next testbed experiments compared the dissemination times and the sync errors of FiGo

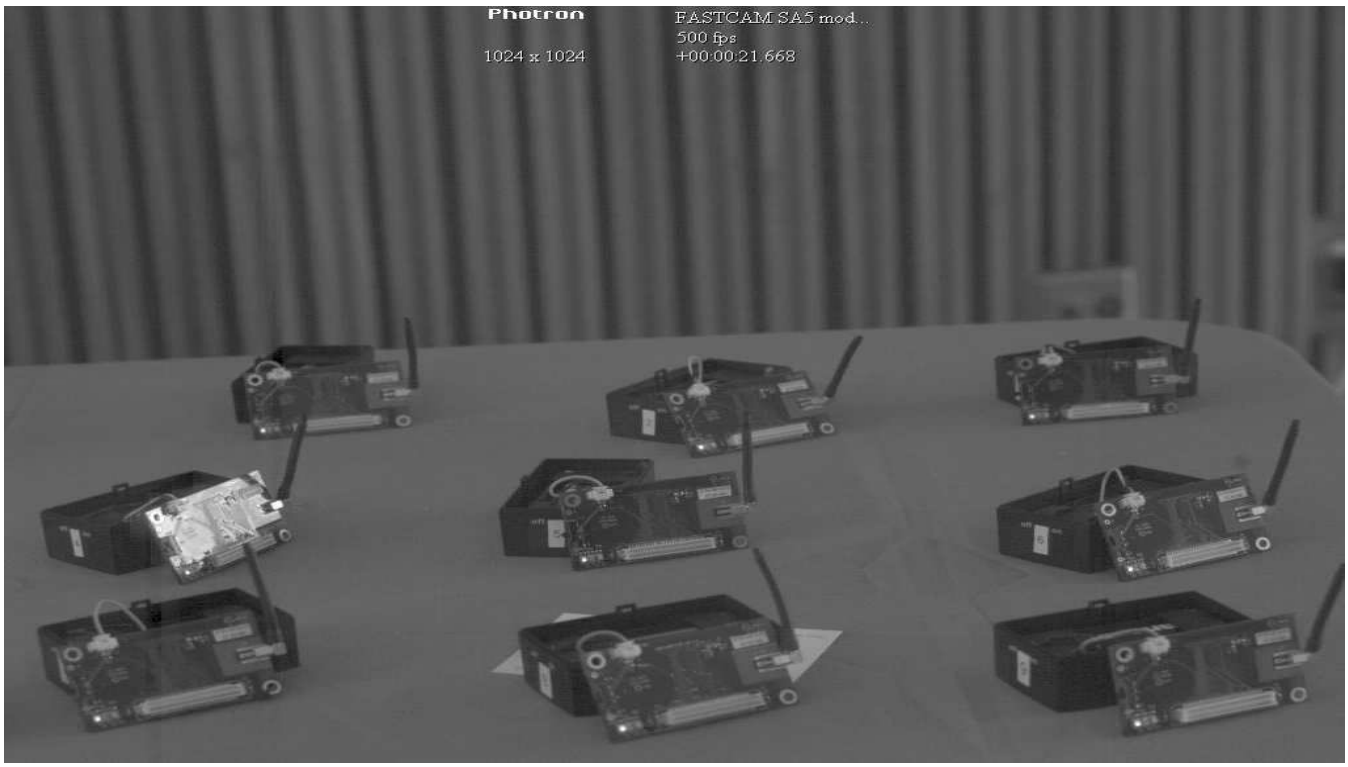

Figure 6.11: Still of a video frame showing the moment that all of the nodes flash within the synchronisation threshold. The start point that I used was when the last node was turned on, which is recorded at time-stamp 00.00.15.760.

with those of FTSP and Drip using the Unified Broadcast(UB) scheme. I evaluate FiGo on its own, as well as FiGo combined with the Greedy Queue (GQ) scheduler. I decided to try a combination of the two schemes because each functions at a different layer. FiGo works as an application/service and works on top of the TinyOS AM message layer. The Greedy Queues scheduler works below the AM message layer, and schedules the protocols access to the radio, as well as the nodes access to the communication medium.

I decided to evaluate against UB because it is an example of a naive protocol combination scheme. UB combines all broadcast traffic. I wanted to evaluate this method of message combination against intelligent message combination. In my case the protocols combined use similar bio-inspired protocols which treat information in the same way, and so lend themselves to combination.

A very important point to make here is that I am not evaluating UB using all of its features. UB gives the ability to a protocol to have its messages sent immediately, instead of waiting until

Protocol	Avg. Dissem. Time	Std. Deviation Time	Avg. Messages Used
Drip-FTSP-CTP with UB	27.5sec	3.52sec	268,752
FiGo	22.03sec	6.52sec	138,967
FiGo with Greedy Queues	24.28sec	5.09sec	138,912

Table 6.5: A comparison of dissemination times on the Indriya testbed with different protocols.

Protocol	Avg. Sync Error	Std. Deviation Sync Errors
Drip-FTSP-CTP with UB	923.09ms	126.73ms
FiGo	55.20ms	11.93ms
FiGo with Greedy Queues	111.26ms	61.42ms

Table 6.6: A comparison of sync errors on the Indriya testbed with different protocols.

the broadcast buffer is full. This particular feature is of importance to FTSP which functions better if it can use this feature. I omitted the use of this feature because I wanted to evaluate the idea of the intelligent combination of protocols against just naively combining them.

The first set of results are summarised in Table 6.5. They show that FiGo without GQ uses 49% less messages than FTSP and Drip when they combine their messages using UB. A very similar result can be seen when FiGo is being scheduled by the GQ scheduler. The vast difference in these results can be attributed in part to the fact that UB combines only broadcast traffic. FiGo combines the control and dissemination information. This means that the same tasks can be performed using FiGo for significantly fewer messages than just naively combining broadcast information.

This result confirms that intelligently combining the messages of two protocols will give good savings in communication overhead when compared with naive message combination. The energy cost of each message can be roughly calculated as $0.64mW$ per message(based on the CC2420 radio transceiver). Table 6.5 shows a savings of roughly 130,000 messages over an hour run, or $277mW$. This goes a long way in prolonging the lifetime of a WSN.

The dissemination time results also showed an improvement with the use of FiGo. FTSP and Drip on UB had the slowest average dissemination time. The dissemination time of FiGo without the GQ scheduler was 19% faster than FTSP, Drip and UB. FiGo combined with GQ was 11% faster than FTSP, Drip and UB. Part of the slow performance of FTSP, Drip and UB

has to be attributed to the fact that the Trickle Timer, which gave Drip better performance in table 6.4 is being negatively affected by having to wait until the UB broadcast buffer is full before being sent. This mitigates the benefit of the Trickle Timer's sliding announcement window. Regardless of problems caused by UB, these results again show the benefits in using intelligent message combination using bio-inspired protocols over simple, naive message combination.

My final experiments looked at synchronisation using FTPS, Drip and UB; FiGo; and FiGo with the GQ scheduler. The results shown in Table 6.6 indicate that FTSP fails when combined with Drip and UB. This failure is a direct result of not using the UB feature to send the FTSP packets immediately. This failure should not be seen as a failure of UB, but as a failure of the notion of naively combining protocol traffic. This problem also affects, to a lesser extent, FiGo when used with the GQ scheduler. During the dissemination periods, the high data requirement of the dissemination protocol causes it to be scheduled in preference to the synchronisation protocol. The result is a larger average synchronisation error, and a larger standard deviation of the synchronisation errors than FiGo with out using the GQ scheduler. FiGo on its own has the smallest average synchronisation error and standard deviation of the synchronisation errors.

These results clearly show that offering multiple services through the use of bio-inspired protocols is possible, and is better than naively combining the communication of non-related protocols like the most commonly cited WSN management systems. These results also show that the combination of the two approaches, queue-length scheduling and combining epidemic algorithms to provide multiple WSN services, needs more work before they can be fully combined.

6.6 Conclusion

In this chapter I presented FiGo, and through it have shown that biological computation can be used to offer multiple services to a WSN with a low communication overhead and in a stable and robust way. The computational result here was that all nodes have the same value for a

given variable after communication with local neighbours. Because both this result and the averaging result required by my synchronisation algorithm use information in the same way due to their use of a bio-inspire swarm intelligence protocol, the messages could be intelligently combined. These services can be provided with a low communication overhead, thereby offering a solution to the WSN management problem of needing to communicate as much system information as possible while using as little communication as possible. I demonstrated a savings of approximately 50% of the messages required to disseminate information to the entire network over current approaches. This further confirms my argument that biological computation can be used to efficiently offer multiple services to a WSN, and help turn a group of sensor nodes into a system.

Throughout this thesis I have been using a smoke monitoring application to illustrate some details about the requirements of an environmental monitoring system. This is the type of system which would use FiGo. FiGo itself has been used as a distributed WSN operating system for several actual deployments. It has provided services to a temperature sensing application, a demo rainfall measuring application, and was used to underpin experimental work done with mobile data sinks. As well as these actual deployments, FiGo has been the operating system used in several remote testbed experiments to test different schedulers. Although based on experimental algorithms, FiGo is a real WSN distributed operating system which has seen real use outside of its own evaluation.

Chapter 7

Summary and Conclusion

Wireless sensor networks offer great promise in the new discoveries that they will enable. Phenomenon all around us, like the movement of smoke in a bar full of people, which are currently difficult to model or measure, will reveal themselves at a greater detail than ever before. But, there are still challenges.

In order for my example smoke monitoring application to produce the results I require, the wireless sensor nodes need a WSN operating system to enable them to work together as a coherent system. In order to work together the nodes need to perform a variety of management functions such as synchronisation, data routing, and command dissemination. There are many obstacles involved in the provision of all of these functions simultaneously. The greatest of these problems is that the nodes need to intercommunicate in order to organise into a coherent system. This is a problem because wireless sensor nodes have limited power resources, and communication is the largest consumer of power. I refer to this as the WSN management problem. WSN nodes need to communicate to form a system but, due to energy constraints, must communicate as little as possible.

The hypothesis of this work is that the WSN management problem is best solved through the efficient self-management and self-organisation of the sensor nodes. The self-management and self-organisation processes can be done efficiently, and in some cases optimally, as the result of biological computation realised through the use of bio-inspired protocols.

In the course of this thesis I have posited that biological systems like ant colonies or fireflies are capable of self-organising without the use of a central controller by performing biological computation. Each and every member of the system holds state variables. Every member will share the value of its variable with all of its immediate neighbours. The value of the local variable is updated based on a function using the values of the variables received from their neighbours. Through the repeated observation of their neighbour's state, and the subsequent adjustment of their own based on their observations, all of the members eventually agree on the value of the state. When the values of all of the states have converged, the computation is done, and the system has organised itself.

This is done by treating a collection of wireless sensor nodes as a computer in its own right. Bio-inspired swarm intelligence protocols enable biological computation which can be used to make WSN self-manage, and provide services to a user. This works because WSN systems share several traits in common with biological systems, such as the way they use information. There are, however, a number of challenges with the use of bio-inspired protocols for WSN use.

The first challenge is that bio-inspired algorithms require large amounts of communication. The natural systems which provide the inspiration for these algorithms can eat and create their own energy. WSN nodes cannot, and are constrained by the limited energy stored in their batteries. I need to approximate the desirable aspects of a bio-inspired protocol in an energy efficient way.

The next challenge is the need to schedule multiple protocols, each with its own communication requirement. A solution is required where one service does not inhibit or interfere with the functioning of the other. The distributed nature of WSN requires a decentralised solution to this problem.

The fact that Bio-inspired swarm intelligence algorithms tend to offer a single service is the final challenge to their use to manage WSNs. As well as scheduling, there is the need to find a way to make the algorithms more flexible so that many services can be offered to an application in a way that is resource efficient and dependable.

What I have shown in this thesis is evidence in support of my hypothesis that biological com-

putation can be used to enable self-management and self-organisation in a resource efficient way, a way of solving my WSN management problem.

In chapter 4 I proposed and evaluated a scheduler to manage the communication of pre-existing protocols. I did this using a bio-inspired protocol which uses only the queue lengths of local neighbours to manage both a node's access to the communication medium, and a protocol's access to the radio. The greedy queue approach uses only local information to do local processing. The emergent result of the local processing is optimal throughput. This optimality is only guaranteed while the collective communication of all of the nodes does not exceed the network's channel capacity.

A limitation to my scheduler was that it only managed broadcast protocols. The approach was fine for management type services like dissemination, but does not include data routing. Both of the routing protocols I examined, CTP and BCP, use broadcast messages to maintain their route information. My scheduler showed an improvement over both the default TinyOS round-robin scheduler, as well as the Unified Broadcast layer and Fair Waiting Protocol when evaluated at a single-hop. An improvement was also seen for multi-hop communication with dissemination time, but message latency caused by the queue-based approach caused the FTSP synchronisation protocol to fail.

Chapter 5 looked at ways to reduce the communication requirements of bio-inspired protocols. I used synchronisation based on epidemic propagation to explore the optimisation potential of bio-inspired protocols. I found that communication could be reduced by almost 80% of previously proposed bio-inspired synchronisation protocols. I also observed the added benefit that reducing communication reduced the time for the protocol to converge.

Our synchronisation protocol very clearly demonstrated that acceptable performance can be expected from a bio-inspired approach for application area such as environmental monitoring. However, centralised architectures are still superior in some areas of performance, for instance when a very small synchronisation error is required, and the risk of loosing the central node can be tolerated. I added the ability to create a global time-stamp in a purely decentralised way. However, FTSP is capable of much more accurate time stamps than my approach, and

suffers far less from a multi-hop environment. I am confident that a distributed notion of skew can be calculated to improve the distributed time-stamp accuracy to make it closer to FTSP's, but at the moment FTSP is still the best protocol for time-stamps with microsecond accuracy.

Finally in chapter 6 I presented the WSN distributed operating system FiGo. Through it I further explored the ability of bio-inspired protocols to combine their communication in an intelligent way that would reduce overall communication without sacrificing protocol performance. This was possible because the protocols I used performed biological computations whose results were the services I required. Biological computations use information in a similar way, and so could be intelligently combined. This allowed us to provide multiple services without increasing message overhead. Testbed results showed that I could reduce communication overhead costs by 40% when compared to the default information dissemination mechanism Drip used by Deluge in the TinyOS library.

Next, I give a list of the contributions of this thesis.

7.1 Contributions

1. My first contribution is to add to the discussion of the use of biological computation through the use of bio-inspired protocols to manage large networks of loosely coupled, low resource, unreliable compute nodes. I recognise in this thesis that making management decisions the results of distributed computation are robust to failure and efficient with respect to communication. I have provided functioning protocols to prove that this approach is feasible, and efficient.

2. More specifically, I presented a bio-inspired cross-layer scheduler which uses only local information to allow multiple, unrelated service protocols to function on the same network without interfering with one another. This protocol determines which protocol an individual node should use, and which node should gain access to the communication medium first. The resulting protocol was shown to be throughput optimal and outperformed the standard round-robin scheduler currently in use by 35% while operating in the network's capacity region.

3. I presented an epidemic synchronisation protocol as another example of a swarm-intelligence based algorithm for WSN management. My protocol was completely decentralised, and could synchronise both events and create a global time-stamp. I showed that my protocol could use less than 80% of the communication of other similar decentralised synchronisation protocols.

4. The final bio-inspired WSN service protocol example was to use the same epidemic algorithm as my synchronisation protocol to provide more services, without greatly increasing communication overhead. A dissemination protocol was presented to illustrate how the use of the same protocol could intelligently combine communication overheads to offer more services while maintaining very low communication costs. I also demonstrated that the performance of the combined protocol was better than when multiple unrelated service protocols were combined together.

5. I have presented FiGo, a WSN distributed operating system. Figo was inspired by problems encountered in actual WSN deployments. Figo has been used as the base for several WSN deployments and as a platform to do experiments.

7.2 Conclusion and Future Work

This work has just begun to scratch the surface of the use of biological computation to enable the distributed control of distributed systems. Now that I have established the usefulness of biological computation in WSNs, I can see several obvious paths for further enquiry. The first is to build upon FiGo, and add data routing to the services it offers. This leads to the generalisation this approach of system management and biological computation to other distributed systems. There exists room to further enhance the performance of biological computation, with respect to speed of convergence and message overhead. Another route is to use WSNs as a platform to explore the behaviour other self-organising algorithms from the field of complexity science.

FiGo's obvious next step is to add data collection and routing to the services it offers. RPL [TED10] currently uses Trickle to disseminate network information to enable data collection. I

have shown in my experiments in section 4 that under a high data collection communication load Trickle may fail. This would affect the ability of RPL to adapt to network change, and would cause it to fail at high data rates. The intelligent combination of network information overhead and use of a protocol scheduler could mitigate this problem. I could also investigate other bio-inspired swarm intelligence protocols for the actual collection of data which would also intelligently combine with the current services offered by FiGo.

There are many other distributed computer systems that may benefit from distributed, biological style self-organisation. Mobile phones could use this form of organisation to enable discovery, synchronisation, and data management for near-field phone-to-phone applications. Office networks could use it to manage portable devices in a scalable way without having to increase the fixed server infrastructure. This can make make the platforms mentioned above behave like large distributed computers in there own right. This may open up new possibilities and applications, as well as allow exploration with new computing paradigms.

There is also further work to be done by exploring and combining other bio-inspired protocols for system management. I have almost exclusively focused on epidemic protocols. There are other bio-inspired protocols which will also combine well with epidemic protocols without incurring a great communication overhead. Particle Swarm Optimisation algorithms and flocking algorithms could be used to localise parts of the network, or create network partitions. Gradient based algorithms could be used to create an information routing protocol in tandem with epidemic dissemination.

There are improvements to be made on the performance of the bio-inspired algorithms. The foremost is to find ways to reduce the message complexity of epidemic protocols. Some work has been done on making the discovery of new data in a network more efficient, but to reduce the messages needed to propagate new data is still an open question. Another area of optimisation is to enhance the accuracy of global timestamps. This would most likely involve the creation of a global notion of skew, and including that in the synchronisation protocol.

However, the real fascinating questions are about what further applications can be produced by using biological computation. The WSN management layer gives the ability to support

further types of complex behaviour. As was mentioned in the background chapter, I have only experimented with the simplest form of emergence. Using Wolfram's four classes of emergent behaviour, I have only explored the first class, where all nodes converge to the same value with out any global control. The fourth class of behaviours are capable of computation. It would be interesting to experiment with protocols which could exhibit these types of complex behaviours, and see what kind of systems could be created. It may be the only way to create truly self adapting systems.

WSNs are a fascinating technology. They can both reveal complex phenomenon, as well as benefit from it. The WSN systems management problem that I address in this thesis is solved by the use of bio-inspired swarm intelligence algorithms to perform biological computation to arrive at a solution. Here I have shown that letting the members of a distributed system decide how to manage themselves is a real possibility. It may also be best approach in the face of scarce resources and uncertainty.

Glossary

Term	Description
Bio-Inspired Computing	The study of the forms of distributed computation performed by biological systems as observed in nature.
Intelligent Combination	The combination of communication and the processing of the information of several different algorithms into a single algorithm due to the fact that the different algorithms process information in the same way.
WSN Management Problem	The problem embodied by the fact that WSN nodes can not afford to communicate very much due to their finite energy resources and that communication is the largest consumer of energy, but that WSN system management requires constant communication between the nodes and any controllers in order to share system information.

Bibliography

[AR05] A. Arora and R. Ramnath. Exscal: Elements of an extreme scale wireless sensor network. In *RTCSA*, pages 102–108, 2005.

[BAM07a] M. Breza, R. Anthony, and J. McCann. Quality-of-context driven autonomicity. *Proceedings of the Second International Workshop on Engineering Emergence in Decentralised Autonomic Systems EEDAS 2007*, 2007.

[BAM07b] M. Breza, R. Anthony, and J. McCann. Scalable and efficient sensor network self-configuration in bioans. *Proceedings of the First IEEE International Conference on Self-Adaptive and Self-Organizing Systems SASO 2007*, July 2007.

[BCD+05] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han. Mantis os: An embedded multithreaded operating system for wireless micro sensor platforms. *Mobile Networks and Applications*, 10(4):563–579, 2005.

[BDWL10] A. Bachir, M. Dohler, T. Watteyne, and K. K. Leung. Mac essentials for wireless sensor networks. *Communications Surveys & Tutorials, IEEE*, 12(2):222–248, 2010.

[BGH+] P. Buonadonna, D. Gay, JM Hellerstein, W. Hong, and S. Madden. TASK: sensor network in a box.

[BJ08] O. Babaoglu and M. Jelasity. Self-* properties through gossiping. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881):3747–3757, 2008.

[BM08] M. Breza and J.A. McCann. Lessons in implementing bio-inspired algorithms on wireless sensor networks. In *Adaptive Hardware and Systems, 2008. AHS'08. NASA/ESA Conference*, pages 271–276. IEEE, 2008.

[BMM+10] M. Breza, P. Martins, J.A. McCann, E. Spyrou, P. Yadav, and S. Yang. Simple solutions for the second decade of wireless sensor networking. In *Proceedings of the 2010 ACM-BCS Visions of Computer Science Conference*, pages 7–17. British Computer Society, 2010.

[BMZN+12] N. Baccour, L. Mottola, M.A. Zu Niga, C.A. Boano, and M. Alves. Radio link quality estimation in wireless sensor networks: a survey. *ACM Trans. Sens. Netw*, 2012.

[BS07] Pruet Boonma and Junichi Suzuki. Bisnet: A biologically-inspired middleware architecture for self-managing wireless sensor networks. *Comput. Netw.*, 51(16):4599–4616, 2007.

[Buc88] J. Buck. Synchronous Rhythmic Flashing of Fireflies. II. *The Quarterly Review of Biology*, 63(3):265–289, 1988.

[BW93] G. Beni and J. Wang. Swarm intelligence in cellular robotic systems. *Robots and Biological Systems: Towards a New Bionics?*, pages 703–712, 1993.

[BYAH06] M. Buettner, G.V. Yee, E. Anderson, and R. Han. X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 307–320. ACM, 2006.

[BYM13] M. Breza, S. Yang, and J. McCann. Multi-protocol scheduling for service provision in WSN. In *SENSORNETS 2013 - Proceedings of the 2nd International Conference on Sensor Networks*. SciTePress, 2013.

[CASH08] Q. Cao, T. Abdelzaher, J. Stankovic, and T. He. The liteos operating system: Towards unix-like abstractions for wireless sensor networks. In *Information Pro-*

cessing in Sensor Networks, 2008. IPSN'08. International Conference on, pages 233–244. IEEE, 2008.

[COKSM05] R. Cardell-Oliver, M. Kranz, K. Smettem, and K. Mayer. A Reactive Soil Moisture Sensor Network: Design and Field Evaluation. *International Journal of Distributed Sensor Networks*, 1(2):149–162, 2005.

[CSR04] J. Carle and D. Simplot-Ryl. Energy efficient area monitoring for sensor networks. *Networks, IEEE Computer*, 37(2):40 – 46, February 2004.

[DBFP09] T. Dang, N. Bulusu, W.C. Feng, and S. Park. Dhv: A code consistency maintenance protocol for multi-hop wireless sensor networks. *Wireless Sensor Networks*, pages 327–342, 2009.

[DBR08] J. Degesys, P. Basu, and J. Redi. Synchronization of strongly pulse-coupled oscillators with refractory periods and random medium access. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 1976–1980. ACM New York, NY, USA, 2008.

[DCA12] M. Doddavenkatappa, M.C. Chan, and AL Ananda. Indriya: A low-cost, 3d wireless sensor network testbed. *Testbeds and Research Infrastructure. Development of Networks and Communities*, pages 302–316, 2012.

[DGH+87] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12, 1987.

[DGV04] A. Dunkels, B. Gronvall, and T. Voigt. Contiki-a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors*, volume 2004, 2004.

[DHJ+06] P. Dutta, J. Hui, J. Jeong, S. Kim, C. Sharp, J. Taneja, G. Tolle, K. Whitehouse, and D. Culler. Trio: enabling sustainable and scalable outdoor wireless

sensor network deployments. *Proceedings of the fifth international conference on Information processing in sensor networks*, pages 407–415, 2006.

[DKS89] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *ACM SIGCOMM Computer Communication Review*, volume 19, pages 1–12. ACM, 1989.

[DMT+11] A. Dunkels, L. Mottola, N. Tsiftes, F. Österlind, J. Eriksson, and N. Finne. The announcement layer: Beacon coordination for the sensornet stack. *Wireless Sensor Networks*, pages 211–226, 2011.

[EHD04] A. El-Hoiydi and J-D Decotignie. Wisemac: an ultra low power mac protocol for the downlink of infrastructure wireless sensor networks. In *Computers and Communications, 2004. Proceedings. ISCC 2004. Ninth International Symposium on*, volume 1, pages 244–251. IEEE, 2004.

[EV06] S.C. Ergen and P. Varaiya. Pedamacs: Power efficient and delay aware medium access protocol for sensor networks. *Mobile Computing, IEEE Transactions on*, 5(7):920–930, 2006.

[FGJL07] R. Fonseca, O. Gnawali, K. Jamieson, and P. Levis. Four-bit wireless link estimation. In *Proceedings of the Sixth Workshop on Hot Topics in Networks (HotNets VI)*, volume 2007, 2007.

[FZTS11] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with glossy. In *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, pages 73–84. IEEE, 2011.

[GFJ+09] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 1–14. ACM, 2009.

[GGS+05] S. Ganeriwal, D. Ganesan, H. Shim, V. Tsiatsis, and M.B. Srivastava. Estimating clock uncertainty for efficient duty-cycling in sensor networks. In *Proceedings of*

the 3rd international conference on Embedded networked sensor systems, pages 130–141. ACM New York, NY, USA, 2005.

[GJP+06] O. Gnawali, K.Y. Jang, J. Paek, M. Vieira, R. Govindan, B. Greenstein, A. Joki, D. Estrin, and E. Kohler. The tenet architecture for tiered sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 153–166. ACM New York, NY, USA, 2006.

[GLvB+03] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, pages 1–11, 2003.

[GM99] D.M. Gordon and N.J. Mehdiabadi. Encounter rate and task allocation in harvester ants. *Behavioral Ecology and Sociobiology*, 45(5):370–377, 1999.

[HC04a] J.W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 81–94. ACM, 2004.

[HC04b] J.W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 81–94, 2004.

[HCB00a] W.R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*, pages 10–pp. IEEE, 2000.

[HCB00b] WR Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*, page 10, 2000.

[HJK11] M.T. Hansen, R. Jurdak, and B. Kusy. Unified broadcast in sensor networks. 2011.

[HKS+05] C.C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava. A dynamic operating system for sensor nodes. In *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pages 163–176. ACM, 2005.

[HL07] G Halkes and Koen Langendoen. Crankshaft: An energy-efficient mac-protocol for dense wireless sensor networks. *Wireless Sensor Networks*, pages 228–244, 2007.

[HMCP04] WB Heinzelman, AL Murphy, HS Carvalho, and MA Perillo. Middleware to support sensor network applications. *Network, IEEE*, 18(1):6–14, 2004.

[HS05] Y.W. Hong and A. Scaglione. A scalable synchronization protocol for large scale sensor networks and its applications. *Selected Areas in Communications, IEEE Journal on*, 23(5):1085–1099, 2005.

[ICKJL09] J. Il Choi, M.A. Kazandjieva, M. Jain, and P. Levis. The case for a network protocol isolation layer. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 267–280. ACM, 2009.

[IVSV06] K. Iwanicki, M. Van Steen, and S. Voulgaris. Gossip-based clock synchronization for large decentralized systems. *Self-Managed Networks, Systems, and Services*, pages 28–42, 2006.

[Jel11] M. Jelasity. Gossip. In Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, and Anthony Karageorgos, editors, *Self-organising Software*, Natural Computing Series, pages 139–162. Springer Berlin Heidelberg, 2011.

[JMB05] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems (TOCS)*, 23(3):219–252, 2005.

[KDG03] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, pages 482–491. IEEE, 2003.

[KM07] K. Kredo and P. Mohapatra. Medium access control in wireless sensor networks. *Computer Networks*, 51(4):961–994, 2007.

[KMAB10] M.Z. Khan, M. Madjid, B. Askwith, and F. Bouhafs. A fault-tolerant network management architecture for wireless sensor networks. *PGNet 2010*, 2010.

[KTDH+11] J.G. Ko, A. Terzis, S. Dawson-Haggerty, D.E. Culler, J.W. Hui, and P. Levis. Connecting low-power and lossy networks to the internet. *Communications Magazine, IEEE*, 49(4):96–101, 2011.

[L+03] P.A. Levis et al. *Trickle: A Self Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks*. Computer Science Division, University of California, 2003.

[Lan86] C.G. Langton. Studying artificial life with cellular automata. *Physica D: Nonlinear Phenomena*, 22(1-3):120–149, 1986.

[LBC+08] P. Levis, E. Brewer, D. Culler, D. Gay, S. Madden, N. Patel, J. Polastre, S. Shenker, R. Szewczyk, and A. Woo. The emergence of a networking primitive in wireless sensor networks. 2008.

[LBV06] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 8–pp. IEEE, 2006.

[LCL07] H. Lee, A. Cerpa, and P. Levis. Improving wireless simulation through noise modeling. In *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, pages 21–30. IEEE, 2007.

[LDCO06a] W. L. Lee, A. Datta, and R. Cardell-Oliver. *Handbook of Mobile Ad Hoc and Pervasive Communications*, chapter Network Management in Wireless Sensor Networks. American Scientific Publishers, USA, 2006.

[LDCO06b] W. L. Lee, A. Datta, and R. Cardell-Oliver. Winms: Wireless sensor network-management system, an adaptive policy-based management for wireless sensor networks tech. rep. uwa-csse-06-001. Technical report, The University of Western Australia, June 2006.

[LLWC03] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: accurate and scalable simulation of entire tinyOS applications. *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 126–137, 2003.

[LMP+05] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, et al. TinyOS: An Operating System for Sensor Networks. *Ambient Intelligence*, pages 115–148, 2005.

[LN90] K. Lindgren and M.G. Nordahl. Universal computation in simple one-dimensional cellular automata. *Complex Systems*, 4(3):299–318, 1990.

[LSS06] X. Lin, N.B. Shroff, and R. Srikant. A tutorial on cross-layer optimization in wireless networks. *Selected Areas in Communications, IEEE Journal on*, 24(8):1452–1463, 2006.

[LSZM04] T. Liu, C. Sadler, P. Zhang, and M. Martonosi. Implementing software on resource-constrainted mobile sensors: Experiences with zebranet and impala. *ACM MobiSYS*, 2004.

[MFHH05] S.R. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems (TODS)*, 30(1):122–173, 2005.

[Mit09] M. Mitchell. *Complexity: A guided tour*. Oxford University Press, USA, 2009.

[Mit11] M. Mitchell. Ubiquity symposium: Biological computation. *Ubiquity*, 2011(February):3, 2011.

[MKSL04] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi. The flooding time synchronization protocol. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 39–49. ACM, 2004.

[MLM+] PJ Marron, A. Lachenmann, D. Minder, J. Hahner, R. Sauter, and K. Rothermel. TinyCubus: a flexible and adaptive framework sensor networks. *Wireless Sensor Networks, 2005. Proceeedings of the Second European Workshop on*, pages 278–289.

[MLM+05] P.J. Marrón, A. Lachenmann, D. Minder, M. Gauger, O. Saukh, and K. Rothermel. Management and configuration issues for sensor networks. *International Journal of Network Management*, 15(4):235–253, 2005.

[MOH04] K. Martinez, R. Ong, and J. Hart. Glacsweb: a sensor network for hostile environments. *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, pages 81–87, 2004.

[MS90] R.E. Mirollo and S.H. Strogatz. Synchronization of pulse-coupled biological oscillators. *SIAM J. Appl. Math*, 50(6):1645–1662, 1990.

[MSKG10] S. Moeller, A. Sridharan, B. Krishnamachari, and O. Gnawali. Routing without routes: The backpressure collection protocol. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 279–290. ACM, 2010.

[NTCS99] S.Y. Ni, Y.C. Tseng, Y.S. Chen, and J.P. Sheu. The broadcast storm problem in a mobile ad hoc network. *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 151–162, 1999.

[OZ06] S. Olariu and A.Y. Zomaya. *Handbook Of Bioinspired Algorithms And Applications*. Chapman & Hall/CRC, 2006.

[PHC04] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 95–107. ACM, 2004.

[Pho09] Photron. Photron highspeed cameras, 2009.

[PNMP12] A. Papadopoulos, A. Navarra, J.A. McCann, and C.M. Pinotti. Vibe: An energy efficient routing protocol for dense and mobile sensor networks. *Journal of Network and Computer Applications*, 35(4):1177–1190, 2012.

[PS07] R. Pagliari and A. Scaglione. Design and implementation of a PCO-based protocol for sensor networks. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 387–388. ACM New York, NY, USA, 2007.

[PSC05] J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 364–369. IEEE, 2005.

[QMG10] J. Qiu, P. Mitchell, and D. Grace. Receiver based interference protection for mac protocol in wsns. In *Wireless Communication Systems (ISWCS), 2010 7th International Symposium on*, pages 902–906. IEEE, 2010.

[RGLAO05] V. Rajendran, J.J. Garcia-Luna-Aveces, and K. Obraczka. Energy-efficient, application-aware medium access for sensor networks. In *Mobile Adhoc and Sensor Systems Conference, 2005. IEEE International Conference on*, pages 8–pp. IEEE, 2005.

[RKRK] A.M. Rosenfeld, D. Kusic, W.C. Regli, and J.B. Kopena. A gossip-based synchronization protocol for state consistency in distributed applications.

[ROGLA06] V. Rajendran, K. Obraczka, and J.J. Garcia-Luna-Aceves. Energy-efficient, collision-free medium access control for wireless sensor networks. *Wireless Networks*, 12(1):63–78, 2006.

[RWA+08] I. Rhee, A. Warrier, M. Aia, J. Min, and M.L. Sichitiu. Z-mac: a hybrid mac for wireless sensor networks. *IEEE/ACM Transactions on Networking (TON)*, 16(3):511–524, 2008.

[SBK05] B. Sundararaman, U. Buy, and A.D. Kshemkalyani. Clock synchronization for wireless sensor networks: a survey. *Ad Hoc Networks*, 3(3):281–323, 2005.

[SNMT07] I. Stoianov, L. Nachman, S. Madden, and T. Tokmouline. PIPENET: A Wireless Sensor Network for Pipeline Monitoring. *Proceedings of the Sixth International Conference on Information Processing in Sensor Networks*, April 2007.

[SSW+] C. Sharp, S. Schaffert, A. Woo, N. Sastry, C. Karlof, S. Sastry, and D. Culler. Design and implementation of a sensor network system for vehicle tracking and autonomous interception. *Wireless Sensor Networks, 2005. Proceeedings of the Second European Workshop on*, pages 93–107.

[TAB10] A. Tyrrell, G. Auer, and C. Bettstetter. Emergent slot synchronization in wireless networks. *Mobile Computing, IEEE Transactions on*, 9(5):719–732, 2010.

[TC] G. Tolle and D. Culler. Design of an application-cooperative management system for wireless sensor networks. *Wireless Sensor Networks, 2005. Proceeedings of the Second European Workshop on*, pages 121–132.

[TED10] N. Tsiftes, J. Eriksson, and A. Dunkels. Low-power wireless ipv6 routing with contikirpl. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 406–407. ACM, 2010.

[TGC06] A.S. Tanenbaum, C. Gamage, and B. Crispo. Taking sensor networks from the lab to the jungle. *Computer*, 39(8):98–100, 2006.

[UC10] M.B. Uddin and C. Castelluccia. Toward clock skew based wireless sensor node services. In *Wireless Internet Conference (WICON), 2010 The 5th Annual ICST*, pages 1–9. IEEE, 2010.

[WASW05] G. Werner-Allen, P. Swieskowski, and M. Welsh. MoteLab: a wireless sensor network testbed. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 483–488, 2005.

[WATP+05] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal. Firefly-inspired sensor network synchronicity with realistic radio effects. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 142–153. ACM New York, NY, USA, 2005.

[Win12] T. Winter. Rpl: Ipv6 routing protocol for low-power and lossy networks. 2012.

[WLP+02] I. Wokoma, I. Liabotis, O. Prnjat, L. Sacks, and I. Marshall. A weakly coupled adaptive gossip protocol for application level active networks. In *Policies for Distributed Systems and Networks, 2002. Proceedings. Third International Workshop on*, pages 244–247, 2002.

[WM04] N. Wakamiya and M. Murata. Scalable and robust scheme for data fusion in sensor networks. In *Proceedings of International Workshop on Biologically Inspired Approaches to Advanced Information Technology (Bio-ADIT)*, pages 112–127, 2004.

[Wol06] S. Wolfram. Cellular automata and complexity. 2006.

[WTC03] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 14–27. ACM, 2003.

[YHE02] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1567–1576. IEEE, 2002.

[YM11] P. Yadav and J.A. McCann. Ebs: decentralised slot synchronisation for broadcast messaging for low-power wireless embedded systems. In *Proceedings of the 5th International Conference on Communication System Software and Middleware*, page 9. ACM, 2011.

[YT08] J. Yu and O. Tirkkonen. Self-Organized Synchronization in Wireless Network. In *Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems-Volume 00*, pages 329–338. IEEE Computer Society Washington, DC, USA, 2008.

[Zim80] H. Zimmermann. Osi reference model–the iso model of architecture for open systems interconnection. *Communications, IEEE Transactions on*, 28(4):425–432, 1980.