

School of Computer Science and Software Engineering  
Monash University

Digital Systems Honours (1200), Clayton Campus

Literature Review - Semester 2, 2003

*Reconfigurable Sensor Networks*

Chris Elliott 12856975

Charles Greif and Nandita Bhattacharjee

## Contents

<b>1</b>	<b>Networked Sensor Characteristics</b>	<b>3</b>
<b>2</b>	<b>Hardware Organisation</b>	<b>4</b>
<b>3</b>	<b>Tiny Operating System</b>	<b>4</b>
<b>4</b>	<b>Self Routing Wireless Network</b>	<b>7</b>
<b>5</b>	<b>Network Sensor Applications</b>	<b>8</b>
<b>6</b>	<b>Future Work</b>	<b>9</b>
<b>7</b>	<b>Conclusion</b>	<b>9</b>

## Abstract

Wireless sensor nodes has come to be an emerging area of study for many groups around the world. This study was funded by Intel and the Department of Defense in America in 1995 and was called *SmartDust*. It uses the idea of processing, sensing, communications and power in one chip (or node) and was mainly developed at the University of California in Berkeley. The idea for the motes is to have a number of them dropped into an environment and gather data which is sent back to a base. This is where the *SmartDust* name originated. The sensor nodes were named motes by the people at Berkeley. Mainly using "off-the-shelf-components" these motes have been created and by using an operating system designed for the motes, they can now be purchased for development of applications. In section one, networked sensor characteristics will be discussed to give a background on the topic. Section two discusses the hardware used to build a mote while section three looks at the Operating System used to develop applications for the motes. How the motes are networked is looked at in section four then examples of applications are given in section five. Finally future work and a conclusion is given in section five and six respectively.

## 1 Networked Sensor Characteristics

The characteristics of a networked sensor system determine its design. These characteristics include size, low power consumption, concurrent operation, software portability and robustness. The motes in a wireless sensor system are of a small size so as to be fairly inconspicuous in the environment they are placed in. This physical requirement raises the problem of power consumption. Typically a mote will use a small battery as a power source and needs to preserve as much power as possible to function as long as it can. As the motes don't have a large amount of memory on them, information must be moved from mote to mote quickly and efficiently so as to use as less power as possible and to relay information gathered before more information is found. Concurrent operation will help preserve the battery power by performing a number of tasks quickly[2] and by doing this, utilizing power resources.

The motes in a network will usually be of a large number and expected to function correctly while being unattended. Therefore enhancing the reliability of individual devices is essential. This can be achieved by accepting individual device failures and factoring them in to an application[2]. The networked motes will usually be application specific, rather than general purpose, and only have the hardware needed for the application. As a wide range of applications are possible, software needed for an application should

be portable across different applications. This means common jobs like sending and receiving information should not have to be reprogrammed for different applications.

## 2 Hardware Organisation

A number of motes with variations in hardware have been used as studies for various areas of research into wireless sensor nodes[1][11][2]. Although using different hardware, the mote still performs the same function. Some of the recently designed motes have become much smaller physically. Some physical sizes as low as one cubic inch[7]. For future applications physical size is said to be paramount[7]. Habitat monitoring has been done successfully on an island[9] with motes named Mica[3]. These motes have a larger physical size of 50mm x 38mm x 13mm. The motes were successful in this application and for this reason, a new version of the Mica mote (Mica2) will be used.

The main microcontroller on the Mica2 mote is an Atmel ATmega 128L low power controller, running at 3.0V. The processor needs to be low power because of the inherent power constraints of a networked sensor. It is an 8-bit microcontroller running at 4MHz with 128Kbytes of flash memory to run software and 512Kbytes of memory for data storage. Included in the microcontroller is an internal 8 channel, 10 bit analog to digital converter[3]. There are a number of interfaces to the processor which allows for connection to a variety of peripherals. The processor integrates a set of timers and counters which can be configured to generate interrupts at regular time intervals[2]. There are three LED's arranged on a mote which can be used to represent outputs connected through I/O ports.

Wireless transmission is performed with a radio transceiver operating at 916MHz with an outdoor range of 150 metres[12]. The Mica2 mote is powered by 2 AA batteries and when the mote is used in a power save mode, these batteries can last for up to a year[12]. The I/O interface consists of a 51-pin expansion connector designed to interface a number of different sensor boards. The sensors found on these boards usually include light, temperature, acoustic, magnetic, acceleration and microphone. A basic block diagram of the Mica2 mote can be seen in figure 1.

## 3 Tiny Operating System

The Tiny Operating System (TinyOS) was developed to act as a generic development environment which allows specialized applications to be con-

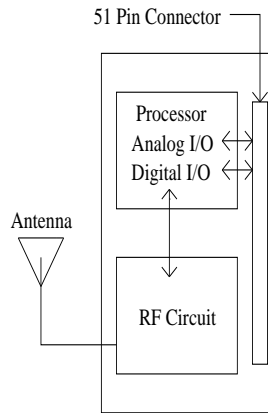


Figure 1: Block Diagram of Mote

structured from a number of common functions[4]. TinyOS is an event based operating system which allows high amounts of concurrent operation in small amounts of memory while utilizing processor resources to preserve power[2]. The operating system is small enough that it can be run from the 128Kbyte of flash memory found on the Mica2 mote.

A complete system configuration consist of a tiny scheduler and a number of components. A component has four interrelated parts: a set of command handlers, a set of event handlers, an encapsulated fixed size frame and a bundle of tasks. Tasks, commands and handlers execute in the context of a frame and operate on its state[2]. A block diagram representing a component can be seen in figure 2.

To facilitate modularity, each component declares the commands it uses and the events it signals[2]. The composition process creates a hierarchy of components where higher level components issue commands to lower level components and lower level components signal events to the higher level components[2]. Physical hardware represents the lowest level of components.

Since the components describe both the resources they provide and the resources they require, connecting them together is simple. The signatures of events and commands provided by another component are matched together to do this. The communication across the components takes the form of a function call, which has a low overhead[2]. Components can be split into three categories: hardware abstractions, synthetic hardware and high level software components. Hardware abstraction components map physical hardware into the component model. Synthetic hardware components

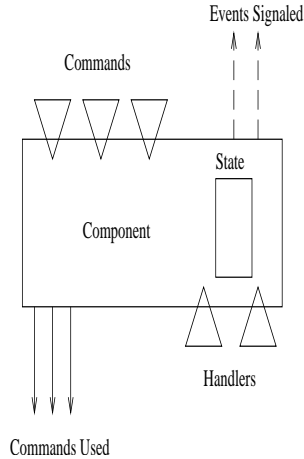


Figure 2: Block Diagram of a Component

simulate the behaviour of advanced hardware. The high level software components perform control, routing and all data transformations[2].

Commands are non-blocking requests made to lower level components[2]. A command will store request parameters into its frame and conditionally post a task for later execution. It may also invoke lower commands, but it must not wait for long or indeterminate latency actions to take place. A command must provide feedback to its caller by returning status indicating whether it was successful or not[2].

Event handlers are invoked to deal with hardware events, either directly or indirectly. The lowest level components have handlers connected directly to hardware interrupts. An event handler can store information into its frame, post tasks, signal higher level events or call lower level commands. A hardware event triggers processing that moves upwards through events and can turn downwards through commands. To avoid cycles in the command and event hierarchy, commands cant signal events. Commands and events are intended to perform a small, fixed amount of work. This occurs within the context of their components frame[2].

Tasks perform the primary work. Tasks can be pre-empted by events but usually run to completion. Tasks can call lower level commands, signal higher level events and schedule other tasks within a component. Tasks allow TinyOS to simulate concurrent operation within each component since they execute asynchronously with respect to events[2].

The task scheduler is a First-In-First-Out queue[2]. When the queue is

empty, the scheduler will put the processor to sleep but leave the peripherals operating so that they may wake up the system. This will help preserve battery power. Once the queue is empty, another task can be scheduled only as a result of an event. Therefore, there is no need for the scheduler to wake up until a hardware event triggers activity[2].

To implement code in TinyOS, the C programming language is used. By using the C compiler (AVRGCC), code written to be implemented on the motes can be compiled into the correct format for the processor found on the motes to function. The component interface is written in a *.comp* file where the declarations for a component are made. This includes the commands that a component accepts, events that the component signals, commands that the component uses and the events that the component handles. The component implementation is written in a *.c* file where functions that implement the interface to the motes is developed. A *.desc* file is also created so that a graph of components can be tied together[6].

## 4 Self Routing Wireless Network

To communicate, the motes use wireless transmission through a radio and *ad-hoc* routing schemes which are necessary as when motes are deployed, there is no organized connectivity before deployment. This type of scheme will still meet the needs of an application as a network topology will be discovered when the motes are set up in an environment. GAaRP (Geographic Addressing and Routing Protocol) is one such scheme that was developed[1]. This scheme used GPS (Global Positioning System) receivers as additional hardware to the motes. The routing protocol involves initial acquisition of a GPS position, followed by a RF (radio frequency) signal message used to trigger communication with modules in RF range. Once a module negotiates and establishes a valid route, the module announces this new route to other modules within RF range. This procedure is repeated in a radial fashion until the entire network is configured relative to a home position. This scheme however, isnt required to arrange the sensor nodes in a self routing network. This routing scheme was developed by Corr and Okino[1] as they decided that information gathered by the sensors should be location dependant as the motes may change location due to environmental changes. This idea would only be necessary if information found needed to be from certain locations.

Similar schemes have been developed without needing to use GPS and in terms of my project, this type of scheme is not necessary. Without additional hardware to be adapted to the motes, a similar routing scheme can be

developed just using components found in TinyOS[5]. A simple scheme takes advantage of the fact that radio communication uses a broadcast medium[8]. The scheme involves one sensor to be used as a basestation which is connected to a host PC and all communication either starts from this basestation or moves toward this basestation. The basestation will broadcast a message asking the other motes to organise into a routing tree. In the message that the basestation broadcasts, it will specify its own ID and its level which is its distance from the basestation and of course will be zero. Any mote which hears this message will assign its own level which will be the level in the message plus one, if its current level is not already less than or equal to the level in the message. It will also choose the sender of the message as its parent, through which it will route messages to the basestation. Each of the motes will then rebroadcast the routing message, inserting their own ID and levels. This routing message floods down the tree in this way until all motes have a parent and a level. Motes that hear a number of parents will select one arbitrarily.

These routing messages are periodically broadcast from the basestation, so that the topology of the motes can be continually updated. This helps the motes to adapt to environmental changes causing them to move location or if a mote fails. When a mote wishes to send a message to the basestation, it sends the message to its parent and the same situation continues until the basestation receives the message. This scheme may not be workable in some specific applications but it is a simple way of getting the motes to communicate.

## 5 Network Sensor Applications

Networked sensors have a large potential to be used in a number of commercial, defense and research applications. This is due to their physical size, wireless communication and relatively low cost. Commercial applications like air conditioning systems may one day use this technology. As the technology gets cheaper, the motes may be a cost effective option to monitor room and building conditions to control these types of systems. The defense applications require that the motes be of a very small size so that large areas can be monitored and that the motes go undetected. A project called Palms[10] was undertaken where a number of these motes were dropped from an unmanned aerial vehicle to monitor movements on a road. The motes organized themselves into a network, detected vehicle movements and sent this information back to the basestation for analysis.

This type of defense application could be used in many research areas

where habitats are monitored. One such application was developed on Great Duck Island[9] in Maine, USA. The application involved monitoring nesting burrows of birds on the island. Thirty-Two motes were deployed into this habitat and over a four week period, temperature readings from the burrows were observed. In the results of this application, it was calculated that the motes would be able to operate for another six months[9]. This timeline means that a large amount of data can be obtained by using these motes and because the data gathered is relayed back to a basestation off the island, valuable time can be used to process the data rather than go on to the island and collect it. This type of research application is what my research will be based on.

## 6 Future Work

As stated in section two, future applications will require motes to be of a small physical, it is said to be paramount[7]. This may be the case for defense applications, requiring motes to go undetected or so small that they are hard to find. It has been shown that larger motes can be used for research[9] and this is where more research can be applied. Developing, testing and analysis of large numbers of motes deployed into an area would be beneficial for future applications where small physical motes will someday be used.

## 7 Conclusion

Much research has been done on developing, testing and programming wireless sensor nodes. Current research is concentrating on how small the motes can be made[7] and some applications have been developed to show how the motes can be used[9]. Some research into reconfiguring the motes[1] has been done but usually reconfiguration is proven through changing network topologies. In my research I will show that the motes can be reconfigured while performing an application so that the functionality of the application can be improved. To do this, I will use the developed hardware discussed in section two and get the nodes to communicate as described in section three. This will be done by using TinyOS and the components developed for this operating system. The idea's behind the Great Duck Island Project[9] will be used as a basis for my research.

## References

- [1] M.G. Corr and C.M. Okino. Networking reconfigurable smart sensors. In *SPIE*, California, 2000.
- [2] Jason Hill and David Culler. System architecture directions for networked sensors. In *Ninth International Conference on Architectural Support for Programming Languages and Operating systems*, Cambridge, MA, 2000.
- [3] Jason Hill and David Culler. A wireless embedded sensor architecture for system-level optimization. Technical report, University of California, Berkeley CA, 2002.
- [4] Jason Hill, David Culler, and Philip Buonadonna. Active message communication for tiny networked sensors. Technical report, Electrical Engineering and Computer Sciences, University of California, Berkeley, CA, 2001.
- [5] Jason Hill, David Culler, and Philip Buonadonna. A network-centric approach to embedded software for tiny devices. In *EMSOFT*, page 16, California, 2001.
- [6] Jason Hill, David Culler, Robert Szewczyk, and Alec Woo. Tinyos tutorial. unpublished, Website <http://www.tinyos.millennium.berkeley.edu>, 2001.
- [7] J.M. Kahn, R.H. Katz, and K.S.J. Pister. Emerging challenges: Mobile networking for smart dust. Technical report, Electrical Engineering and Computer Sciences, University of California, Berkeley, CA, 2000.
- [8] Samuel Madden, David Culler, Robert Szewczyk, and Micheal J Franklin. Supporting aggregate queries over ad-hoc wireless sensor networks. Technical report, University of California, Berkeley CA, 2001.
- [9] Alan Mainwaring, Joseph Polastre, Robert Szewczy, Dvaid Culler, and John Anderson. Wireless sensor networks for habitat monitoring. In *WSNA*, Atlanta Georgia, 2002.
- [10] K.S.J. Pister. Palms fixed/mobile experiment. unpublished, submitted on website <http://robotics.eecs.berkeley.edu/pister/29Palms0103/>, 2001.
- [11] Jason Reidy and Robert Szewczyk. Power and control in networked sensors. Technical report, University of California, Berkeley CA, 2000.
- [12] Crossbow technology. Mica2 wireless measurement system. unpublished, Mote Data Sheet from Crossbow Technology, 2002.