

# FAULT TOLERANCE IN SENSOR NETWORKS USING SELF-DIAGNOSING SENSOR NODES

S Harte<sup>1</sup>, A Rahman<sup>1</sup>, K M Razeeb<sup>2</sup>

<sup>1</sup> University of Limerick, Ireland

<sup>2</sup> Tyndall National Institute, Ireland

## ABSTRACT

This paper describes a method of introducing a level of fault tolerance into wireless sensor networks by monitoring the status of each wireless sensor node. We focus on the detection of physical malfunctions, caused by impacts or incorrect orientation. We design a flexible circuit using accelerometers that acts as a sensing layer around a node, which will be capable of sensing the physical condition of a node. Software analysis is performed on the raw data from the accelerometers to determine the orientation of the node and to detect impacts. The information from the analysis enables us to assess the damage probability. After detecting that a node is not healthy, the effect this has on the whole sensor network is looked at. We explore the changes that can be required such as treating sensor readings as invalid. At the network level, the routing protocol should be made aware of faulty nodes to ensure that faulty nodes are routed around.

## 1. INTRODUCTION

Wireless sensor networks are a recent development. They are possible because of progress in other areas such as Micro-Electro-Mechanical System (MEMS) technology, which is being used in sensors, along with improvements in transceivers and microcontrollers. This progress has led to the possibility of combining sensors, RF transceivers and processing capability into small, low-power nodes. A wireless sensor network is a network made up of a number of such nodes. Currently networks with hundreds of nodes are possible, but future networks should be capable of orders of magnitude greater numbers than this, as the nodes get smaller and more power efficient.

There is a variety of application areas of Wireless sensor networks. An example application is habitat monitoring. In this case, a wireless sensor network is deployed over an area such as a forest, as carried out by Mainwaring et al (1) or even a volcano, as by Werner-Allen et al (2). Numerous measurements can then be made such as temperature, humidity, seismic activity etc. These measurements can be stored and analysed, and action can be taken if needed. The advantage of wireless sensor networks is that they are less intrusive than other methods. Another possible use of wireless sensor networks is in creating intelligent environments.

This involves using sensors and actuators to create responsive environments. A simple example of this is turning on a light automatically as a person enters a room. More advanced possibilities can be developed by Srivastava et al (3), where a smart learning environment is created.

Figure 1 shows the general architecture of wireless sensor networks. There are a number of different configurations possible but most follow a similar design. Each node has a radio frequency transceiver, a number of sensors, and a microprocessor. These components should be chosen with energy consumption in mind, particularly the radio transceiver, as it uses most of the energy, as shown by Raghunathan et al (4). When nodes are deployed, they will be running on batteries, so it is important that they use as little energy as possible, to be able to remain operational for longer.

Each of the nodes in the network communicates using a low-power radio transceiver. The exception is the base node, which is wired directly to a PC. For the PC to send data to the network requires sending the data to the base node, which then forwards the data over its radio. For the PC to receive data from the network, the data must be passed through the base node. The PC allows the collected data to be stored in a database, and can allow for remote monitoring if connected to the Internet.

For large networks, it is unlikely that the base node is within transmission range of all the other nodes. Even if this is not true, it can be more energy efficient to forward data over a number of nodes. This results in the need for a multi-hop routing protocol. A number of different protocols have been proposed, however one of the more commonly implemented solutions involves arranging the nodes into a tree structure. An overview of this is given by Levis et al (5). In the tree structure, when a node transmits data, a number of nodes may receive the data, but only the parent node of the sender will forward on the data.

This paper is concerned with adding a layer of fault tolerance capabilities to sensor networks. Any one of the components on a wireless node can become faulty for a number of reasons. For example a sensor reading could be incorrect because of the physical condition e.g. if a node has a light sensor, and the node rolls over so that the light sensor is underneath the node. Alternatively, the electrical connection between a sensor and a node could be damaged due to accidental or malicious impacts. This paper focuses on detecting such

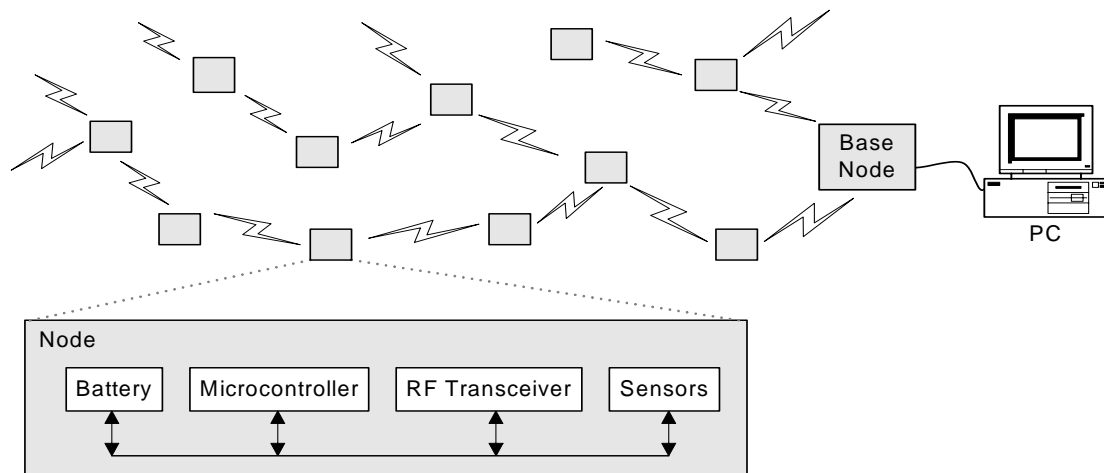


Figure 1: Wireless sensor network architecture

physical faults; however it may be extended to other types of damage or malfunction, e.g. using temperature sensors to detect if a node is operating outside its range.

There are other types of errors that cannot be detected by this technique: the analogue-to-digital conversion can introduce aliasing errors, incorrect programming of the nodes will cause problems, and batteries will eventually fail on a node.

This paper is split into numbered sections. Section 2 deals with the hardware and supporting software design of the sensing layer. Analysis of the data to introduce a level of fault tolerance is discussed in Section 3. Section 4 covers the future work that will be done, including implementation of the ideas presented in this paper.

## 2. SENSING LAYER DESIGN

To detect physical faults requires the use of hardware and a software interface. The hardware consists of a sensing layer that wraps around the node. The software reads the sensors, and transmits the data to the PC, through the base node.

The wireless sensor node used for this work is the DSYS25 node that is presented by Barroso et al (6). This node was developed at the Tyndall National Institute (formerly NMRC), Ireland. It is unique due to its modular design. Modules for the node are designed as a 25mm x 25mm printed circuit board. These connect together through high-density 80-way, 40-way and 20-way connectors. One of these nodes is shown in Figure 2, with four different modules. The DSYS25 node has an Atmel ATmega128L microcontroller and a Nordic nRF2401 RF transceiver, provides wireless communication between nodes. Other modules containing thermistors, humidity sensors, or any other small sensors can be connected easily.

This node was chosen mainly for its compact size, and high level of connectivity between layers.

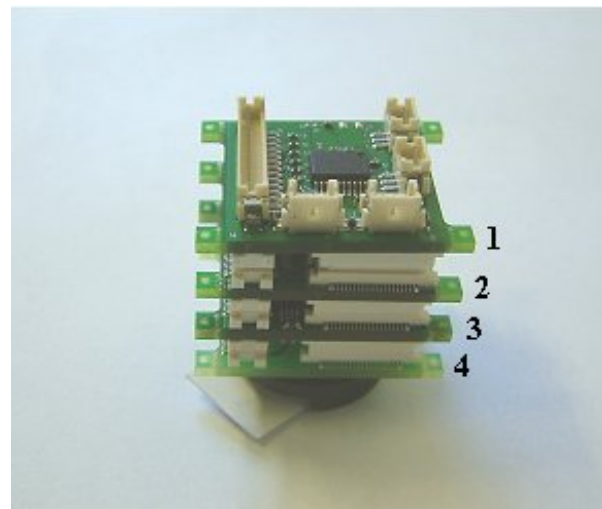


Figure 2: DSYS25 node.

### 2.1 Circuit Design

The design of a physical sensing layer is a novel approach to fault detection in wireless nodes. The layer will be capable of detecting the orientation of the node and impacts on the node. The sensing layer consists of a number of miniature accelerometers mounted on a flexible PCB. The PCB will envelop the node, and connect the accelerometers to the node. The multiple accelerometers allows for multiple readings to be made for increased accuracy. It also introduces some redundancy into the design if some of the accelerometers are damaged.

For this project, the main constraint on the accelerometers is that they must use little power, and have a small footprint. The Analog Devices ADXL320 accelerometer fitted these requirements well. It

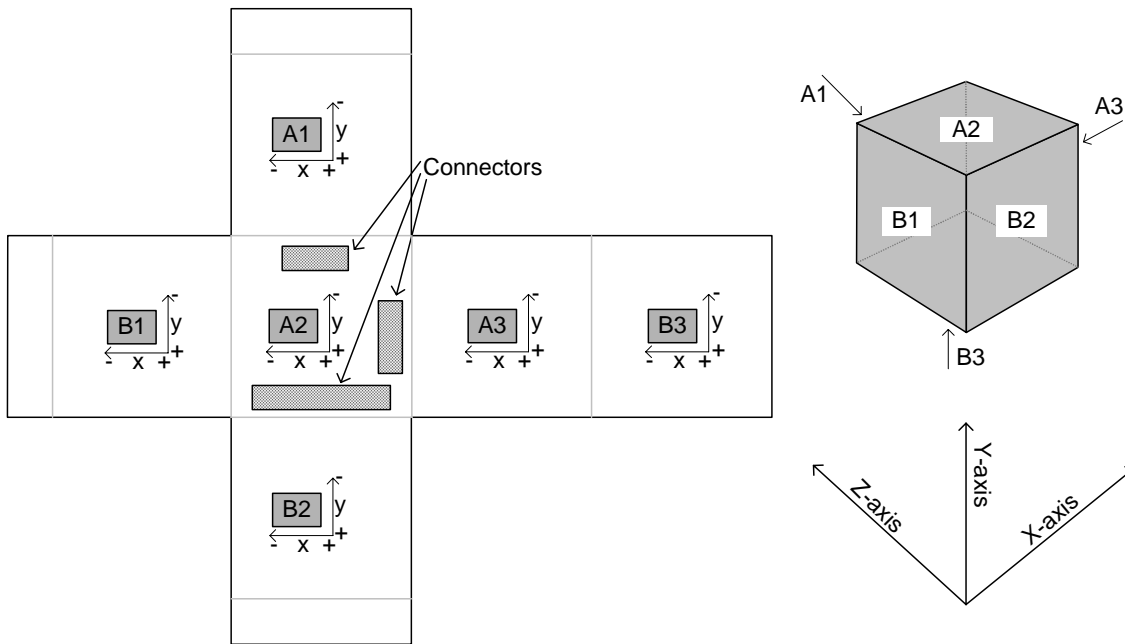


Figure 3: Sensing layer: unfolded (components are on bottom side); and folded

measures 4mm x 4mm x 1.45mm, and uses 0.45mA of current, according to the ADXL datasheet (7). It is a 2-axis accelerometer and its analogue output is very straightforward to interface to a microcontroller.

Figure 3 shows a simplified view of the position of the accelerometers and connectors on the circuit. The accelerometers and connectors are on the bottom side of the circuit as viewed. Each of the sensors has been designated a name: A1, A2 etc. The orientation of each sensor is shown by the direction of the x and y output of each of the sensor. In addition, Figure 3 also how the circuit folds into a cube shape. The actual circuit has rounded corners, as sharp corners can lead to tracks on the PCB breaking. An X-axis, Y-axis, and Z-axis for the cube are also defined.

As each sensor has two analogue outputs, there are 12 analogue outputs altogether. This created a problem as the ATmega128L microcontroller only has eight ADC inputs (ADC0 to ADC7). To solve this, the sensors were divided into two groups: {A1, A2, A3} and {B1, B2, B3}. Corresponding elements from each group are connected to the same ADC inputs on the microcontroller. For example, sensor A1 and sensor B1 both have their x output connected to ADC0 and their y output connected to ADC1. This will not cause a conflict, as only one of the groups will be powered on at any time. The sensors were arranged so that each group has no parallel sensors. This ensures that each group covers all three axes.

## 2.2 Software Interface

The TinyOS operating system is used as a platform for the development of software to perform sensor readings and transmit readings at each node. TinyOS is an

operating system specifically designed for wireless sensor nodes, and so as a very small footprint, and is energy-aware. It is written in the nesC programming language, which was also designed for wireless sensor nodes. TinyOS is an event-based operating system. It waits in a low-power state for an event to occur, then it processes the event and returns to a low-power state. TinyOS has been ported to the DSYS25 node. This was done as part of the D-Systems Project (8).

As TinyOS already provides components for doing analogue-to-digital conversions, and components for radio transmission, the unique code needed for this project is quite small. This code is written in, like TinyOS itself, the nesC programming language, which is introduced by Gay et al (9) and defined in (10).

The code for this project makes use of the three TinyOS component. ADCC is an interface to the physical ADC on the microcontroller. The GenericComm component is used for communications. TimerC is a component that controls the timers on the microcontroller. Figure 4 shows the components that are unique to this work (shaded components) and how they connect to existing components.

When the code is running on the nodes, the sensors are regularly sampled, and the readings are transmitted back to the PC.

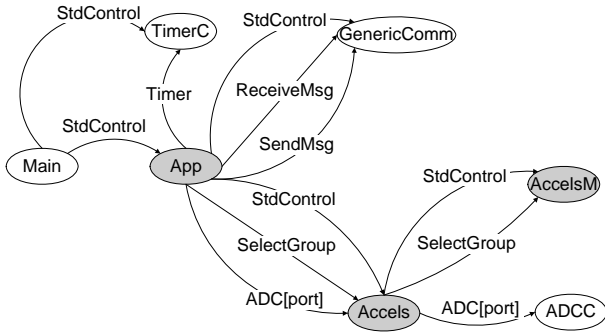


Figure 4: TinyOS components used

### 3. DATA ANALYSIS

The data from the sensors has two distinct uses. It must be able to identify impacts and determine the orientation of the node. These are distinct because identifying impacts involves looking for significant changes in the readings from the sensors, whereas determining the orientation needs the readings to be stable.

Before any analysis of the readings can be carried out, the readings are converted into units of g. To do this requires knowing the sensitivity and the 0g value of the accelerometer. These values are given as 0.174V/g and 1.5V, for a supply voltage of 3V in (7). The details of the ADC must also be known. The ADC on the Atmega128L microcontroller gives 10-bit data, and a 3V reference voltage is used. Using this information gives the following formula to convert raw ADC output values into measurement of acceleration in units of g.

$$acceleration = \frac{ADCoutput \times 3}{2^{10} - 1} - 1.5$$

$$\Rightarrow acceleration = \frac{ADCoutput}{59.223} - 8.6207$$

#### 3.1 Determining Orientation

We will look first at how we determine the orientation of the nodes. In this scenario, all the sensor readings will be below 1g, and the last few sensor readings should have been stable. This guarantees that the sensor is at rest.

This section refers to each of the sensors by the designators given in Figure 3. The x and y outputs of the sensors are referred to by A1[x] and A1[y].

The acceleration for the X-axis, Y-axis, and Z-axis (as shown in Figure 3) can then be found by averaging the

readings from the relevant sensors, as shown in the equations below:

$$X_{accel} = \frac{B2[x] - B3[x] + A1[x] + A2[x]}{4}$$

$$Y_{accel} = \frac{A1[y] + B1[x] - B2[y] - A3[x]}{4}$$

$$Z_{accel} = \frac{-B1[y] - A2[y] - A3[y] - B3[y]}{4}$$

These values can be converted to angles by using the inverse sine function, as shown in (7).

#### 3.2 Detecting Impacts

In this case, the accelerometers will be giving readings greater than 1g, and the readings will not be stable. The data from each output of each sensor can be combined as before, giving values for Xaccel, Yaccel, and Zaccel. The magnitude of the acceleration at any time will be given by:

$$magnitude = \sqrt{X_{accel}^2 + Y_{accel}^2 + Z_{accel}^2}$$

In order to detect an impact has occurred, the magnitude of acceleration is monitored over time. If the magnitude of acceleration goes over a threshold value, it can be said that an impact has occurred. The threshold value can be decided by experiment. An example of this is given in Figure 5. It can be seen that the magnitude goes over the threshold repeatedly for a short period of time. This could correlate with the sensor being hit.

Once the impact has been detected, it is signalled visually on the PC. The magnitude of the impact can then be used to assess if it is likely that any of the components on the node have been damaged. This requires knowledge of the components being used.

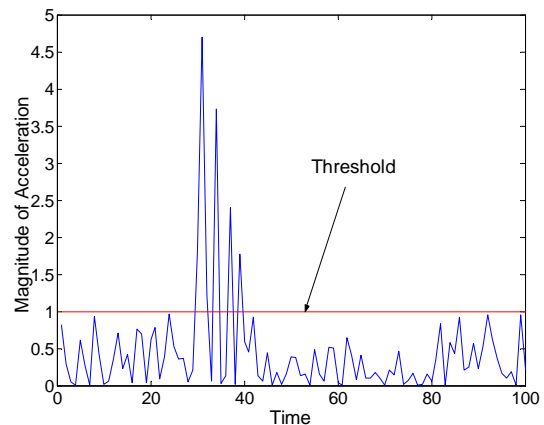


Figure 5: Acceleration magnitude changing over time

#### 4. FUTURE WORK

The current software is designed to periodically sample the accelerometers, and transmit the readings to the PC. An improvement over this would be to provide two methods of getting the data to the PC. The first way is that the PC can send a request for the data. The request is sent throughout the wireless network, and is received by every node. When a node receives a request, it responds with the latest data for each of the six accelerometers. Secondly, the nodes are constantly sampling the sensors. If the readings go above a certain threshold value, the readings will be sent to the PC. The first method of getting information should prove useful for getting data to use for calculating the orientation of the node. The second way is using the processing power on the nodes to detect impacts. This will result in less data being transmitted, and so will reduce energy consumption.

The software on the node can be also be modified so that when an impact is detected, the node can run some diagnostics to check if any components have been damaged. This will consist of checking that all the sensors and the transceiver still respond to the microcontroller.

Currently tests have been simulated, but as part of this work, tests will be carried out using physical nodes. This will allow us to test the capabilities of the sensing layer in a realistic setting. It will allow for interesting possibilities in how the sensor network is affected by the loss of nodes. When a sensor fails, the routing protocol needs to be aware of this so that the faulty node can be routed around. An efficient method to do this is given by Staddon et al (11).

Another possible direction for this work is modifying the sensing layer to provide a water-tight seal. This would be very useful in sensor nodes being used for habitat monitoring, as explained in Szewczyk et al (12). Care should be taken that the accuracy of some sensors will decrease when enclosed, for example humidity sensors.

#### 5. CONCLUSION

The approach to fault-detection presented here is mainly a hardware approach, and is based on information collected from the node. This is not a complete fault-detection solution: if the node is completely destroyed, the data will never reach the PC and never be analysed. In this case, approaches based on nodes monitoring each other are useful. Such approaches are given in Krishnamachari and Iyengar (13) and in Ruiz et al (14). (13) uses data from nearby nodes to detect if an event has occurred. If the nearby nodes agree, then it is likely

that an event has occurred, whereas if the nearby nodes disagree, it is likely that there is a fault. (14) presents a management architecture for event-driven sensor networks.

The sensing layer can be added to any wireless sensor network using the DSYS25 nodes. It is possible to build a wide range of wireless sensor networks using this node, as different modules can be easily connected to build the required sensor node, with the desired capabilities. For example an FPGA can be added to provide more processing power, or many different sensors or actuators can be connected.

Our approach provides a way to detect interference with a sensor node. This could be accidental e.g. moved by an animal. It could also be malicious, if the sensor node was being used as part of a security system for example. The sensor node may continue to give correct data after being moved or hit, but knowing that the sensor was moved or hit allows the data from that node to be treated with suspicion, until it can be manually verified that the node is ok.

#### ACKNOWLEDGEMENTS

This work was done with support from the Tyndall National Institute, Ireland through the National Access Program (NAP). We are grateful for the assistance given by Brendan O'Flynn at the Tyndall National Institute.

#### REFERENCES

1. Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R., and Anderson, J., 2002, "Wireless Sensor Networks for Habitat Monitoring," Proc. 1st ACM International Workshop on Wireless Sensor Networks and Applications, pp. 88 – 97.
2. Werner-Allen, G., Johnson, J., Ruiz, M., Lees, J., and Welsh, M., 2005, "Monitoring Volcanic Eruptions with a Wireless Sensor Network," Proc. 2nd European Workshop on Wireless Sensor Networks.
3. Srivastava, M., Muntz, R., and Potkonjak, M., 2001, "Smart Kindergarten: Sensor-based Wireless Networks for Smart Developmental Problem-solving Environments," Proc. 7th annual international conference on Mobile computing and networking, pp. 132 – 138.
4. Raghunathan, V., Schurgers, C., Park, S., and Srivastava M., 2002, "Energy-Aware Wireless Microsensor Networks," IEEE Signal Processing Magazine, vol. 19, no. 2, pp. 40 – 50.

5. Levis, P., Madden, S., Gay, D., Polastre, J., Szewczyk, R., Woo, A., Brewer, E., and Culler, D., 2004, "The Emergence of Networking Abstractions and Techniques in TinyOS," Proc. 1st USENIX/ACM Symposium on Networked Systems Design and Implementation, pp. 1 – 14.
6. Barroso, A., Benson, J., Murphy, T., Roedig, U., Sreenan, C., Barton, J., Bellis, S., O'Flynn, B., and Delaney, K., 2004, "Demo Abstract: The DSYS25 Sensor Platform," Proc. 2nd international conference on Embedded networked sensor systems, pp. 314.
7. Analog Devices, ADXL320 datasheet, <http://www.analog.com/en/prod/0,2877,ADXL320,00.html>
8. The D-Systems Project website, <http://www.cs.ucc.ie/misl/dsystems>
9. Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E. and Culler, D., 2003, "The nesC Language: A Holistic Approach to Networked Embedded Systems," ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 1 – 11.
10. Gay, D., Levis, P., Culler, D., and Brewer, E., "nesC 1.1 Language Reference Manual," <http://nesc.sourceforge.net>
11. Staddon, J., Balfanz D., and Durfee, G., 2002, "Efficient Tracing of Failed Nodes in Sensor Networks," Proc. 1st ACM international workshop on Wireless sensor networks and applications.
12. Szewczyk, R., Polastre, J., Mainwaring, A., and Culler D., 2004, "Lessons Learned From A Sensor Expedition," Proc. 1st European Workshop on Wireless Sensor Networks, pp. 122 -130.
13. Krishnamachari B., and Iyengar S., 2004 "Distributed Bayesian Algorithms for Fault-Tolerant Event Region Detection in Wireless Sensor Networks," IEEE Transactions on Computers, vol. 53, no. 3, pp. 241 – 250.
14. Ruiz, L., Siqueira, I., e Oliveira, L., Wong H., Nogueira, J., and Loureiro, A., 2004 "Fault management in event-driven wireless sensor networks", Proc. 7th ACM international symposium on Modelling, analysis and simulation of wireless and mobile systems, pp. 149 – 146.