Wireless sensor network monitoring using the Simple Network Management Protocol

Leon du T. Steenkamp

Supervisor: Richardt H. Wilkinson
Co-supervisor: Shaun Kaplan
Centre for Instrumentation Research

Submitted in fulfillment of the
requirements for the
degree of Magister Technologiae
in the Department of Electrical Engineering

Cape Peninsula University of Technology
Cape Town

August 16, 2012

Declaration

I, Leon du Toit Steenkamp, declare that the contents of this thesis represent my own unaided work, and that the thesis has not previously been submitted for academic examination towards any qualification. Furthermore, it represents my own opinions and not necessarily those of the Cape Peninsula University of Technology.

Signature of author ....

Cape Town
November 2011

Acknowledgments

I would like to acknowledge the following people and groups who in some way or another contributed towards the completion of this thesis:

- Aan Pa en Ma, vir al die hulp, bystand, aanmoediging en ondersteuning wat julle my gegee het,
- my supervisors Mr Shaun Kaplan and Dr Richardt Wilkinson for their advice and support,
- the Centre for Instrumentation Research for its support and equipment and
- the French South African Institute of Technology for its assistance and allowing me to work on this document while also completing their coursework.

The financial assistance of the National Research Foundation, French South African Institute of Technology and Cape Peninsula University of Technology towards this research is acknowledged. Opinions expressed in this thesis and the conclusions arrived at, are those of the author, and are not necessarily to be attributed to the respective mentioned establishments.

Synopsis

Wireless sensor networks (WSNs) have long been separate networks using non-standard, custom and application specific protocols. The arrival of IPv6 over Low power Wireless Personal Area Network (6LoWPAN) changes this, with the Internet Protocol (IP) being the common protocol between conventional IP networks and Wireless Sensor Networks.

Through the use of 6LoWPAN implementations like *blip*, developed for the WSN operating system TinyOS, it is possible to make use of the advantages that the common IP layer brings. This includes the reuse of techniques, protocols and software that are already developed for, and in use on, conventional IP networks.

One such protocol is the Simple Network Management Protocol (SNMP), which is a User Datagram Protocol (UDP) based protocol used for monitoring and management in conventional IP networks.

This work describes an implementation of the first version of the SNMP in nesC for TinyOS using the *blip* 6LoWPAN implementation. The SNMPv1 software agent implementation requires no modification to the 6LoWPAN stack and responds to standard SNMP get and set requests. This means that standard SNMP software can be used to communicate with the implemented SNMP software agent. Therefore network monitoring packages like Nagios and Cacti can be used to monitor wireless sensor networks.

Relevant background information is discussed, ranging from the division of communication between network nodes into logical layers, to similar work done and the Simple Network Management Protocol. The implementation of the SNMP software agent is examined, along with the anatomy of an SNMP GetRequest message in aid of understanding the structure of SNMP messages.

The SNMP agent implementation is then tested against network monitoring tools like Cacti, Nagios and applications from the Net-SNMP package. The results verify the correct operation of the SNMP software agent and demonstrate the added functionality that the software agent provides.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

6LoWPAN	IPv6 over Low power WPAN
ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules
HTTP	Hypertext Transfer Protocol
IAB	Internet Architecture Board
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPv6	Internet Protocol Version 6
ISO	International Standards Organisation
ITU-T	International Telecommunication Union - Telecommunication Standardisation Sector
LAN	Local Area Network
LPL	Low Power Listen
MAC	Medium Access Control
MIB	Management Information Base
MTU	Maximum Transmission Unit
NMS	Network Monitoring System/Station
OSI	Open Systems Interconnection
PDU	Protocol Data Unit

PHY	Physical Layer
RFC	Request for Comments
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WPAN	Wireless Personal Area Network
WSN	Wireless Sensor Network

Chapter 1

Introduction

Wireless sensor networks (WSNs) are networks of low power devices that use their on-board sensors to cooperatively monitor their surroundings. Even though various real world and test WSNs have been designed and constructed they are not always easy to connect to other networks. WSNs are mostly separate, stand-alone networks that use custom, nonstandard messages. This makes each WSN different from another and difficult to connect without a complex translation gateway device. It also means that for each network task a custom protocol or way of dealing with the task is often devised, where on conventional Internet Protocol (IP) networks standardised protocols and methods already exist for dealing with these tasks, which can often be reused. One important task is network monitoring. Network monitoring in WSNs is not a new activity (Yu, Mokhtar & Merabti, 2006) with various solutions devised to deal with this task. The conventional IP network solution to network monitoring and management is the Simple Network Management Protocol (SNMP).

The current state of affairs in WSNs is similar to the early days of networking before the Internet, where no two networks could easily communicate and were interconnected with difficulty. For conventional networks things started to change with the advent of the Internet Protocol with more and more networks adopting IP. With this common protocol between networks it was possible to find common ground and interconnecting networks became easier. It might be possible for version six of the Internet Protocol (IPv6) with the help of the 6LoWPAN (IPv6 over Low power WPAN) specification to do something similar for WSNs. The 6LoWPAN specification describes a method and protocol for sending IPv6 messages over WSNs. IPv6, through the use of 6LoWPAN, becomes the common protocol between the different WSNs. This protocol is not only common to the WSNs implementing 6LoWPAN, but also to conventional networks using IPv6. 6LoWPAN enabled WSNs can therefore also be easily connected to conventional networks, such as wired Local Area Networks (LANs) or Wireless Local Area Networks

(WLANs), by using much simpler gateway devices than needed before.

The 6LoWPAN specification does not only bring this common protocol to WSNs but also the possibility of using protocols, methods and tools already developed for conventional IP networks on WSNs. One of these protocols is the Simple Network Management Protocol (SNMP). The SNMP is used on conventional IP networks to monitor and manage network nodes.

The objective of this research is to implement and test an SNMP software agent for WSNs that can respond with and receive standard SNMP messages. This SNMP software agent can then be used in the task of network monitoring and management on WSNs.

Some broad research questions that will be considered in the work and answered as formulated in the research methodology section are as follows: What is the SNMP and where is it used? What network monitoring exists in WSNs, specifically SNMP like solutions? What network monitoring packages, for conventional computer networks exist that can be reused in WSNs? Is an SNMP implementation for WSNs possible?

1.1 Research methodology

A literature study on the Simple Network Management Protocol was done and the different versions and aspects of the SNMP were looked at. Papers on network monitoring and management in WSNs were investigated with a focus on SNMP and SNMP-like implementations and methods. The different network monitoring software packages that are generally used on conventional networks, but could also be used to monitor WSNs, were investigated.

The design of an SNMP software agent for wireless sensor network devices was done. Design decisions that were made as to what version to implement and what functionality to include are discussed. The designed SNMP software agent was implemented in nesC for the TinyOS WSN operating system.

The implemented SNMP software agent was tested during and after development for correct operation and functionality. For testing purposes a WSN testbed was created on which some of the testing was done. Testing was done to ensure that the network stack was functioning correctly. The software agent was tested against command line SNMP tools and also tested over a longer period on the testbed using the chosen network monitoring software. The software agent was tested against existing network monitoring software and tools, which was possible because the agent interpreted and responded with standard SNMP messages.

1.2 Significance and contributions of the research

The research done offers a glimpse into the possibilities that an SNMP enabled 6LoWPAN WSN hold. It does so without having to do modifications to the 6LoWPAN stack used and also without the need for a translation gateway. The SNMPv1 software agent using *blip* in TinyOS is also one of the first such implementations, if not the only, available that can operate without the need for an SNMP proxy or translation gateway. *blip* is a 6LoWPAN implementation created by researchers at the University of California at Berkeley for the TinyOS WSN operating system.

This work attempts to show that it is possible, through the use of SNMP and enabled by 6LoWPAN, to use existing network monitoring software developed for conventional networks with WSNs.

The implemented SNMP software agent can be used in future WSN deployments by the Centre for Instrumentation Research (CIR) to facilitate easy network monitoring of deployed WSNs. It would also be possible to use the SNMP software agent to facilitate the gathering of the sensor readings. One such proposed WSN deployment is one to do condition monitoring of high voltage transformers. Through the use of the created software agent, network monitoring can be done using existing software packages and additional development of software for this purpose is not necessary. The software agent therefore has real life application in its use in future deployments to perform the essential function of network monitoring.

A byproduct of the work done is the constructed WSN testbed. The testbed can be used by students and staff for testing and developing WSN applications or for class projects. The testbed can be further extended to include more nodes or improved software.

1.3 Delineation of the research

This work focuses on network monitoring for WSNs using the SNMP without any modifications to the protocol, or need for translation gateways or nodes. It takes into account the limitations of available wireless sensor network node platforms, especially the TelosB mote platform, and also the limitations of current software implementations. The research was done using the TinyOS 2.1.1 operating system for WSN nodes and Ubuntu 9.10 was used as a desktop operating system.

It was not possible to do a complete SNMP software agent implementation due to limitations of the WSN platform, the WSN environment and the large scope of the SNMP, consisting of three versions. Functionality was implemented according to its suitability in WSNs. The large number of possible network monitoring and management server configurations could not be fully explored, but proof of concept setups are shown including a few different software packages. Research did not include aspects of security or optimisation in terms of software agent code size. In this document the terms LoWPAN and WSN are used interchangeably, as are the terms network management and network monitoring.

1.4 Structure of the document

The general structure of the document can be seen in Figure 1.1. This chapter contained a general introduction to the work with methodology and significance being discussed.

The following chapter has an overview and background of the topics and concepts discussed later in the document. These include a brief description of WSNs, layered communications models, hardware and network monitoring software.

The 6LoWPAN specification and SNMP is covered in more detail in chapter three and a closer look is taken at the anatomy of an SNMP message.

The topics shown in Figure 1.1 in the blocks titled *Background* and *6LoWPAN and SNMP* relate to the conceptual layered communications model and each topic is covered in the corresponding section.

Chapter four deals with the implementation of the SNMPv1 agent in TinyOS. The different components and applications created are also explained in more detail.

The second last chapter deals with testing and evaluation of the SNMPv1 agent and presents data gathered using the SNMP over 6LoWPAN. The last chapter is used to conclude the work done and mentions closing remarks such as problems encountered and proposed further work.

Various appendices are included, these contain information and work done that does not fit within the main body of the thesis. More information is given on the testbed created and the set up and configuration of the software used. The TinyOS source code is listed and the PHP graphing scripts are looked at in more detail.

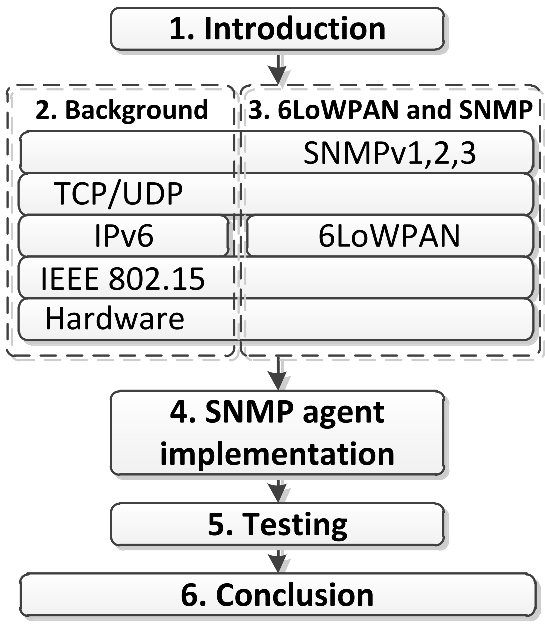


Figure 1.1: Structure of the document

Chapter 2

Background

This chapter discusses background information on topics covered in the work. The following sections give a general introduction to WSNs, the WSN operating systems available and some WSN hardware. The OSI layered communications model is discussed to give perspective on where other protocols used fit in. Different network monitoring software options are discussed along with similar work in the form of WSN specific network monitoring.

2.1 An introduction to wireless sensor networks

A wireless sensor network (WSN) can be described as a network of small, low power devices that cooperatively monitor the space throughout which they are distributed. These devices communicate over a wireless communication medium and typically have a very limited range. They can be fitted with any number of different sensors. The sensor nodes, or motes, often operate from a very limited power supply and for this reason spend most of their time in a sleep state. These sleep intervals can have a duty cycle of as low as five to ten percent, only running for five to ten percent of the day.

When using WSNs there are some difficulties that need to be overcome. One example is the limited power (or energy) available to the devices in the network to power themselves. Another is the radio transceivers used. These are low bit rate, low power, have a limited communication range and typically conform to the IEEE 802.15.4 standard. WSN devices are also restricted in terms of the available memory and computational power.

One of the difficulties caused by using low power radio transceivers, with only a limited communication range, can be solved by employing multihop or mesh routing techniques.

With these techniques nodes can relay messages for other nodes. The range of a single node can be extended using multiple nodes, with messages traversing multiple hops to their destination.

WSNs are an attractive option because they open sensing possibilities that were not previously possible. Using WSNs, sensitive or dangerous environments can be easily monitored and a high spatial density of sensors attained that might not have been possible previously with conventional monitoring.

One case in point is the well known WSN deployment on Great Duck Island, Maine (Mainwaring, Culler, Polastre, Szewczyk & Anderson, 2002; Szewczyk, Mainwaring, Polastre, Anderson & Culler, 2004). Researchers were able to gather data in an area sensitive to human presence due to the nesting birds on the island. Szewczyk *et al.* (2004) stated that the long-term and unattended operation of the WSN enabled the gathering of measurements at a spatial and temporal resolution that would have been impractical with human observers or with sparsely deployed instruments.

A large number of published works are available that show how WSNs can be used in various applications and used to gather large datasets with millions of points sometimes spanning months. A WSN was used to monitor the microclimate in and around a coastal redwood tree, with data being gathered over 44 days and motes placed at two meter intervals in the tree (Tolle, Polastre, Szewczyk, Culler, Turner, Tu, Burgess, Dawson, Buonadonna, Gay & Hong, 2005). A WSN was used to monitor the efficiency of water use in irrigation (McCulloch, McCarthy, Guru, Peng, Hugo & Terhorst, 2008). Researchers were also able to monitor volcanic activity using a WSN (Werner-Allen, Johnson, Ruiz, Lees & Welsh, 2005). Having a successful deployment is however not always a trivial task (Langendoen, Baggio & Visser, 2006).

2.2 Wireless sensor network specific network monitoring

Network monitoring in WSNs is not a new activity and numerous attempts have been made at monitoring the health and performance of a WSN through various means (Yu *et al.*, 2006; Lee, Datta & Cardell-Oliver, 2007). Most of these works do not however employ the network monitoring over an IP enabled WSN (Hsin & Liu, 2006; Tolle & Culler, 2005). The divisions between the different layers in the OSI (Open Systems Interconnection) model are also not that clear in the stacks used. In TinyOS there is a dilution of the boundaries of layers that is partly due to the Active Message Layer used in TinyOS, which promotes the implementation of protocols directly onto Link Layer frames

(Hui, 2008). TinyOS is an event based operating system specifically developed for WSNs and used in this research. This is likely to be true for most WSN operating systems that do not offer higher layer services by default and only implement a rudimentary framing of messages.

Some of the solutions proposed for WSN monitoring and management are based on the SNMP (Simple Network Management Protocol) or implement some concepts found in the SNMP. Most of these works based on the SNMP also do not make use of 6LoWPAN (Jang, Jeong & Choi, 2006; Stefanov, 2008). A number of SNMP based solutions take advantage of the proxy based deployment model, or some form of sub agent. This deployment model sees a proxy type device deployed between the SNMP agent (managed device) and the management station (Vancea & Dobrota, 2007; Jacquot, Chanet, Hou, Diao & Li, 2009). Some SNMP based implementations use this proxy based deployment model to facilitate protocol translation which makes it possible to perform compression of messages (Choi, Kim & Cha, 2009).

A paper from mid 2009 by Choi *et al.* (2009) proposes 6LoWPAN-SNMP. The work describes a protocol that uses SNMP and *b6LoWPAN*, the predecessor to the *blip* 6LoWPAN implementation. 6LoWPAN-SNMP also uses a proxy forwarder on the gateway device. Messages from the gateway device to the WSN nodes are of a compressed SNMP nature.

LiveNCM is one of the protocols that employ an SNMP sub-agent or proxy type deployment model. LiveNCM employs its own protocol between the gateway (housing the SNMP agent and sub-agent) and WSN node (Jacquot *et al.*, 2009). The LiveNCM management tool does also not make use of 6LoWPAN.

Similar approaches are also used in other works (Jang *et al.*, 2006; Vancea & Dobrota, 2007), where some form of proxy or sub-agent is implemented on the gateway device. A non-SNMP protocol is normally used over the WSN from the gateway device to the WSN node.

2.3 Wireless sensor network operating systems

Various operating systems for wireless sensor networks (WSNs) exist, most of these are open source. The following is a short list of some of the available operating systems. The characteristics of interest are support for the TelosB platform and the Internet Protocol.

Contiki (Dunkels, 2010): This OS is actively being developed by a group from academia and industry under the lead of Adam Dunkels from the Swedish Institute of Computer

Science and is targeted at microcontrollers with limited resources. Contiki uses an event driven kernel and is written in the C programming language. It uses 'protothreads' to provide a thread like programming style. It is possible to dynamically load and unload applications at runtime. Contiki does offer support for IP based communication in the forms of uIPv6 and SICSlowpan, a 6LoWPAN implementation.

Hardware platforms available for Contiki include the TelosB and ESB platform. The ESB platform is based around the MSP430 microcontroller from Texas Instruments with a TR1001 low power radio transceiver from RFM. It also has a few sensors on board: temperature, sound, passive IR and vibration. The TelosB platform will be covered in section 2.4.

MANTIS (Han, 2007): The OS developed by the MANTIS group at the University of Colorado at Boulder is written in the C programming language. It is an open source OS for WSNs and is multithreaded. Platform support includes Mica2, MicaZ and Telos nodes. This project does not appear to be very active.

Nano-RK (Nano-RK, 2010): Nano-RK is available under a dual license similar to what is used by Qt from Qt Development Frameworks (formerly Trolltech). The Open Source Edition is available under the GNU General Public License (GPL). Nano-RK is a fully pre-emptive reservation based real-time operating system (RTOS) written in the C programming language. The OS from Carnegie Mellon University supports the FireFly and MicaZ platforms and offers multi-hop networking and various link layer protocols.

LiteOS (LiteOS, 2010): LiteOS is an open source OS supporting the MicaZ and Iris platforms and is written in the C programming language. LiteOS is not event driven and offers a UNIX like experience.

TinyOS (TinyOS, 2010): TinyOS is an open source OS released under the BSD license and designed specifically for WSNs. It supports multiple hardware platforms including the Mica-family, Telos-family and IntelMote2. Development can be done in MS Windows or Linux. TinyOS is an event driven OS that uses a component-based architecture.

TinyOS uses an extension to the C programming language called nesC. The operating system makes use of different components that each have their own interfaces through which it communicates. These different components are wired together to form applications (Gay, Levis, Culler & Brewer, 2003; WEBS, 2004).

Initially developed at the University of California at Berkeley it is currently being actively developed by a community of contributors. Various network protocols have been developed for use with TinyOS, some being: Dissemination, Tymo, Deluge T2 and *blip* (TinyOS, 2010).

TinyOS was chosen as an OS because it had all the functionality needed and there was a functional 6LoWPAN stack available. The candidate also had previous experience with the OS and therefore did not need to spend time learning a new OS. Assistance from peers was also available for the OS and TinyOS supported the hardware available.

2.4 Wireless sensor network hardware

Several companies manufacture WSN hardware. The hardware ranges from modules to be used in other products to complete services or solutions making use of WSN technology. The following is a short list of manufacturers and limited details on their products or services.

Arch Rock (Cisco, 2010a) offers their PhyNet Wireless Sensor Networking Platform and Arch Rock Energy Optimizer products. Each of these products is available in different packages that consist of a varying number of WSN nodes, PhyNet routers and PhyNet servers. The WSN nodes use the TelosB platform. Arch Rock also supplies the needed software and do not offer the ability to reprogram the nodes with custom firmware. Arch Rock was acquired by Cisco in September of 2010 (Cisco, 2010a).

Sensinode (Sensinode Ltd, 2011) also offers complete packaged solutions with no development needed. Hardware can be bought in the form of kits containing NanoSensors and NanoRouters. The NanoRouters come in two variants, one that uses the USB (Universal Serial Bus) and the other Ethernet. Both variants support the IEEE 802.15.4 wireless standard. Sensinode also provides the needed software.

Jennic (Jennic, 2011) provides development kits, wireless microcontrollers and modules. Their product offerings are not the same complete, ready-to-use solutions that Arch Rock and Sensinode offer. Jennic products can be used to implement solutions for wireless sensor network applications. Their wireless microcontrollers and modules make use of the IEEE 802.15.4 standard.

Crossbow (Crossbow, 2011) offers a complete WSN solution for environmental monitoring called eKo. They also manufacture wireless modules, sensor boards and WSN gateways. Crossbow modules come in various platforms including Iris, MicaZ, Mica2, Imite2, Cricket and TelosB. MEMSIC (MEMSIC, 2011) acquired most of Crossbow's WSN business in January 2010.

Sentilla (Sentilla Corporation, 2011) offers solutions based on wireless sensor network technology in the form of the Sentilla Energy Manager. Sentilla also offers the 'Sentilla Perk: Pervasive Computing Kit'. Moteiv, the predecessor of Sentilla, manufactured various modules based on different platforms, one of which was the TelosB.

2.4.1 TelosB

Various WSN platforms exist but due to the choice of WSN OS and the *blip* stack the TelosB hardware platform was chosen. The primary target for the *blip* stack is the Epic platform, which is based on TelosB. The other supported platforms include TelosB, Iris and MicaZ. The Texas Instruments MSP430 based IEEE 802.15.4 compliant TelosB hardware platform was available for use and so additional hardware did not have to be purchased.

Various manufacturers offer WSN nodes based on the TelosB platform, two of which are Crossbow and Maxfor (Figure 2.1). The platform uses a 16-bit MSP430F1611 MCU (Microcontroller Unit) from Texas Instruments and a CC2420 wireless transceiver from Chipcon. It has 48 kB of program Flash, 10 kB of RAM and 1 MB of serial flash external to the MCU. A 12-bit SAR (Successive Approximation Register) ADC (Analogue to Digital Converter) and 12-bit DAC (Digital to Analogue Converter) are also available. The 2.4 GHz, IEEE 802.15.4 radio transceiver offers a data rate of 250 kbps and selectable RF output power, from -24 dBm to 0 dBm. Temperature, relative humidity and visible light sensors are available on the platform. TinyOS offers support for the TelosB platform and the various sensors available on the platform.

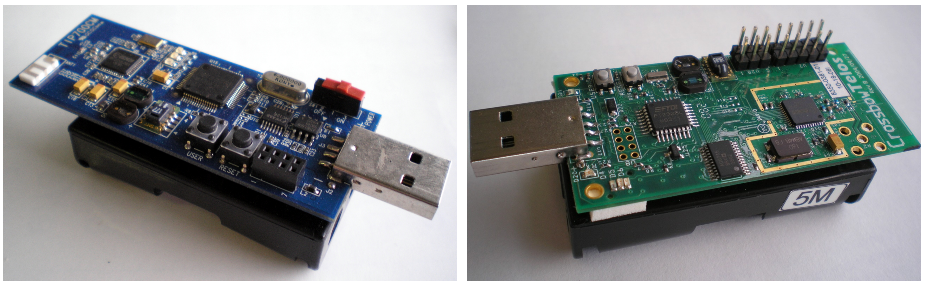


Figure 2.1: TelosB by Maxfor (TIP700CM) *(Left)* and Crossbow *(Right)*

2.5 OSI layered model

An abstract and conceptual model called the Open System Interconnection (OSI) reference model is used to describe how information moves from one application running on a networked device to another. The OSI reference model divides the process of sending a message from one networked device to another into seven layers or seven smaller, more manageable tasks, see Figure 2.2. Grouped together in each layer is a set of similar conceptual network functions. Each layer communicates to the layer immediately above and below it through Service Access Points (SAPs).

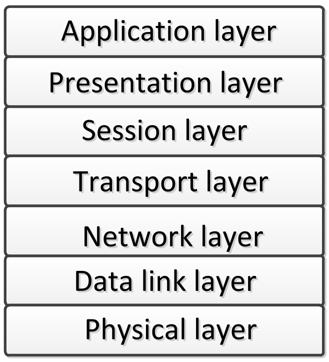


Figure 2.2: OSI Layered Model (Shay, 2004)

A network protocol stack that is developed in this layered manner is easier to maintain and continued development is assisted by the layered approach because protocols can be replaced or improved without affecting the other layers. The stack is also more likely to be reused because new protocols can be developed implementing new techniques or new protocol implementations supporting new hardware can be done.

A brief description of each layer follows (Shay, 2004):

- Physical layer: The Physical layer (PHY) specifies the electrical, mechanical and procedural elements that make communication possible. For example, if a wired medium is used, the Physical layer would contain specifics on the layout, number of pins and voltages used between the communicating devices. If a wireless medium is used, the frequency at which the transceiver operates, channel characteristics and modulation scheme are just some of the details specified by the Physical layer.

- Data link layer: This layer is responsible for controlling the flow of information over the link and also for error detection and correction. The Data link layer divides the incoming and outgoing bits into frames by marking the beginning and end with special bit patterns. The Data link layer is often divided into two sub layers: Logical Link Control (LLC) and Medium Access Control (MAC). The LLC sublayer manages communications of devices over a single link. It also enables multiple higher layer protocols to use the same physical link. The MAC sublayer manages how the device accesses the physical network medium. MAC addresses enable devices to be uniquely identified at the Data link layer.

- Network layer: Network routing is performed at this layer along with message fragmentation and reassembly. A logical addressing scheme is used at the Network layer, which differs from the physical address in that it is logically assigned to the device according to where it is in the network and it is often used in the act of routing. The

Network layer is responsible for establishing end-to-end communication between nodes. An example of a Network layer protocol is the Internet Protocol (IP).

- Transport layer: The Transport layer provides multiplexing, buffering and connection management. Flow control often also takes place at this layer. The Transport layer in general offers two connection modes to the upper layers, connection-orientated or connectionless. When the connection-orientated mode is used, the protocol has to first set up a logical connection between the two protocol implementations before data is sent. The connection is terminated after the transfer of the data is completed. With the connectionless mode, the data is sent as it is received by the Transport layer protocol. The multiplexing service allows several users to use the Transport layer on a single node. Examples of protocols that operate at this level are TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). TCP offers a reliable connection-orientated service that ensures that data is delivered while UDP offers only a best-effort connectionless service.

- Session layer: The protocols in the session layer are responsible for establishing, maintaining and terminating a session between two end users. This layer also provides simplex, half-duplex or full-duplex communication. The Session layer provides the connection between users while the Transport layer provides a connection between nodes.

- Presentation layer: This layer provides encryption and decryption of data and also deals with the presentation of data. It enables different syntax and semantics to be used by higher layers. Data can be converted from a representation understood by the application layer to a network format. This layer would also provide translation between different application data representation formats. Data compression can also be done at this layer.

- Application layer: The Application layer is the closest to the user and directly interacts with the software application. This layer typically is responsible for synchronizing communication, identifying communication partners and determining resource availability. Examples of Application layer protocols are HTTP (Hypertext Transfer Protocol) and SNMP.

A message travels from the application to the top most layer, Application layer, and then travels down each layer onto the physical medium. On the receiving device the message moves from the bottom most layer, the Physical layer, upwards until it reaches the application. Application of this layered approach can be seen in the protocol stack most commonly used on the Internet.

2.6 Internet layered model/protocol suite

The Internet protocol stack, or protocol suite, follows the above-mentioned layered model. In the Internet protocol stack, shown in Figure 2.3, some of the layers are omitted or appear with different names from the traditional OSI model. The Internet model consists of four layers, some texts use three (Padlipsky, 1982) or five (Shay, 2004) but the most common is four (Braden, 1989).

In this model everything below the Internet layer and above the Physical medium is combined into a single layer - referred to as the Link layer, with the host required to implement the communications protocol used to interface with that particular network (Baker, 1995). RFC 1812, edited by Baker (1995) also states that protocols at the Link layer, and by inference the Physical layer, generally fall outside the scope of Internet standardisation.

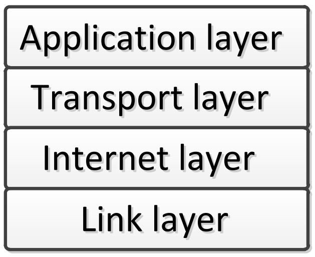


Figure 2.3: Internet Layered Model (Braden, 1989)

Using this layered approach was very successful because it enabled new protocols to be added to the protocol suite and the original protocols could be updated and improved without the need to change protocols that exist at the other layers. One example of this is the development of IPv6 to eventually replace IPv4. Protocols like SNMP could also be developed and added to the protocol suite and go through the evolution from SNMPv1 to SNMPv3.

2.7 The IEEE 802.15.4 protocol suite

IEEE 802.15.4 describes the Physical and Medium Access Control layers most used in WSNs and falls within what is specified under the Internet model as the Link layer.

IEEE 802.15 (Heile, 2011) is the denotation for the IEEE's working group for Wireless Personal Area Networks (WPAN). The working group has several task groups focusing on different areas of WPANs. These include Task Group 1 through to 7. IEEE 802.15

is a sister working group to IEEE 802.11, which is responsible for Wireless Local Area Network (WLAN) standards. Both the 802.15 and 802.11 IEEE standards fall under the IEEE 802 LAN/MAN Standards Committee.

Task Group 1 (Gifford, 2004) was responsible for the IEEE 802.15.1 standard and subsequent amendments to the standard that was based on the Bluetooth V1.1 Foundation specifications. The specification details the MAC and PHY layers.

Task Group 2 (IEEE, 2004) addressed the issue of coexistence of WPANs (IEEE 802.15) and Wireless Local Area Networks (WLAN IEEE 802.11) operating in the unlicensed frequency bands.

Task Group 3 (Barr, 2009) focused on high rate WPANs. The IEEE 802.15.3 standard document specifies MAC and PHY layers with data rates of between 11 and 55 Mbps.

Task Group 4 (Kinney, 2010) produced IEEE 802.15.4 documents describing MAC and PHY layers for low rate WPANs.

Task Groups 5, 6 and 7 are responsible for mesh networking of WPANs (Lee, 2011), short range Body Area Networks (BAN) (Astrin, 2011) and Visible Light Communication (VLC) (Won, 2011).

Other Interest Groups include the Terahertz Interest Group (IGthz) (Kurner, 2003) and the Wireless Next Generation Standing Committee (SCwng) (Kinney, 2011).

The IEEE 802.15.4 standard (IEEE, 2006) for Low Rate WPANs (LR-WPANs) is the set of specifications for MAC and PHY layers most commonly used for communication in WSNs. WPANs are used to communicate over short distances with little or no infrastructure making it possible to implement inexpensive, low power solutions for a range of devices. The standard defines the use of a contention based Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA) medium access mechanism with support for peer-to-peer and star topologies. For time critical messages the optional super frame structure can be used to allocate guaranteed time slots to those nodes. The 2006 revision of the standard offers a selection of PHY layers:

- 868/915 MHz Direct Sequence Spread Spectrum (DSSS) with Binary Phase-Shift Keying (BPSK) as modulation scheme,
- 868/915 MHz DSSS PHY with Offset Quadrature Phase-Shift Keying (O-QPSK),
- 868/915 MHz Paralleled Sequence Spread Spectrum (PSSS) PHY with BPSK and Amplitude Shift Keying (ASK) and
- 2450 MHz DSSS with O-QPSK.

The standard makes provision for 16 channels in the 2450 MHz band, 30 channels in the 915 MHz band and 3 channels in the 868 MHz band.

The 2450 MHz PHY supports the highest over-the-air data rate at 250k bits per second. IEEE 802.15.4 allows for 16-bit short or 64-bit extended addresses, fully acknowledged protocols, low power consumption, energy detection in the channel and link quality indication (LQI). The standard makes provision for two different types of devices on the network, full-function devices (FFD) and reduced-function devices (RFD). A RFD is intended for simple tasks and can therefore be implemented with minimal resources. A RFD can only be connected to a single FFD at any given time. An FFD can however fulfil three roles in the network. It can serve as a Personal Area Network (PAN) coordinator, a coordinator or a device. FFDs can connect to other FFDs and RFDs. There is backwards compatibility between the 2003 and 2006 revision of the IEEE 802.15.4 standard.

IEEE 802.15.4a-2007 is an amendment to IEEE 802.15.4-2006 describing alternate PHY layers that could be used. The alternative PHY layers provide improved robustness, communication range and mobility over IEEE 802.15.4-2006. The layers also make provision for precision ranging with an accuracy of one meter or better. The alternate PHYs comprise of an Ultra-Wide Band (UWB) PHY with frequencies ranging from 3 - 5 GHz, 6 - 10 GHz and less than 1 GHz; and Chirp Spread Spectrum (CSS) PHY at 2.450 GHz.

Task Group 4b was tasked to add enhancements and clarifications to IEEE 802.15.4 including the consideration of new frequency allocations, reducing complexity, resolving ambiguities and increasing flexibility in security key usage. The work done by this task group was published as IEEE 802.15.4-2006, which is the current revision of the IEEE 802.15.4 standard. The revision adds to the market applicability of IEEE 802.15.4 and adds improvements uncovered by implementations of IEEE 802.15.4.

IEEE 802.15 Task Group 4c is responsible for adding an amendment to the PHY of the IEEE 802.15.4-2006 standard and the IEEE 802.15.4a-2007 amendment. The PHY amendment is in response to Chinese regulatory changes, which has opened the 314-316 MHz, 430-434 MHz, and 779-787 MHz bands in that country for use in WPANs. The task group has decided that the 779-787 MHz band can be utilised for the IEEE 802.15.4 standard and an MPSK (M-ary phase-shift keying) PHY and O-QPSK (Offset quadrature phase-shift keying) PHY defined. Eight channels were assigned in the 780 MHz band.

Similar to amendments made by Task Group 4c, IEEE Task Group 4d is tasked with amending the IEEE 802.15.4 standard to define a new PHY and MAC to support a new frequency allocation in Japan, 950-956 MHz with 22 channels.

Zigbee builds upon the IEEE 802.15.4 standard and defines communication protocols used in the upper layers of a proprietary WPAN standard. The relationship between the IEEE 802.15.4 standard and Zigbee (Zigbee Foundation, 2010) can be likened to the relationship between IEEE 802.11 and the Wi-Fi Alliance. The Zigbee alliance is a group of companies that pay to be part of the alliance.

The Zigbee alliance also defines Public Application Profiles that contain specific information about the device and information on how the device should interact with the network. These include Smart Energy and Home Automation. In 2009 the Radio Frequency for Consumer Electronics (RF4CE) consortium decided to work with the Zigbee alliance on Zigbee RF4CE. Zigbee RF4CE will be used in remotely controlled consumer electronics, replacing infrared remote controls.

2.8 Internet layer

At the current point in time one of the drivers for adoption of IPv6 (Internet Protocol version 6) is the diminishing available IPv4 address space. IPv6 was designed as the replacement for IPv4 and offers various improvements over its predecessor. The most notable of these is the increase of IP address size from 32 bits in IPv4 to 128 bits in IPv6. This translates to 3,911,873,538,269,506,102 addresses per square meter of the Earth's surface (Hinden, 1996).

IPv6 also offers simplified address auto configuration compared to its predecessor. The header format in IPv6 has been heavily revised and header fields from IPv4 have been dropped or moved to optional headers. This makes the base IPv6 header lighter and reduces the cost of processing the header in the sense that you only 'pay for what you use'. Any extra header fields such as fragmentation, routing, hop-by-hop options or destination options headers are added as needed. This stacked header approach has the benefit of greater flexibility and introducing new options and headers in future becomes as simple as defining an extension header (Deering & Hinden, 1998).

IPv6 requires a link capable of a maximum transmission unit (MTU) of 1280 octets or greater. An octet consists of eight consecutive bits, equivalent to what is today generally understood by the term byte. If a link cannot carry a 1280 octet packet then a layer below IPv6 must provide a fragmentation and reassembly service. Fragmentation in IPv6 only occurs at the source node and not at routers along the packet's path, in contrast to IPv4. The packet is reassembled only once it has reached its destination.

2.9 Transport layer

Two widely used Transport layer protocols currently on the Internet are the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). These two protocols offer widely differing services to the Application layer.

The use of TCP offers a highly reliable host-to-host protocol that is connection oriented and end-to-end reliable. TCP requires that each packet that is sent be acknowledged (ACK) by the recipient. Each packet is also assigned a sequence number, along with the ACK and a checksum. It is used to provide reliable communication between nodes. Lost packets are retransmitted after a timeout interval, out-of-order received packets can be reordered, duplicates eliminated and damaged packets handled accordingly. TCP also provides flow control and congestion control, which along with the reliability offered by TCP require that certain state information about each data stream be held. Flow control is implemented through the use of a sliding window protocol where the window size field in the TCP segment is used to indicate the number of octets a receiver is able to receive. TCP has to first establish a connection with the other protocol implementation before data transfer can begin and when complete the connection should be terminated. (Postel, 1981)

The User Datagram Protocol (UDP) on the other hand only offers best effort connectionless packet delivery using a minimalist protocol mechanism. UDP offers packet delivery with minimal overhead but in doing so offers no guarantee as to delivery or duplicate protection. UDP does have a checksum value and together with source address, destination address and the UDP length gives protection against misrouted datagrams. UDP is used by protocols such as the Simple Network Management Protocol (SNMP) and the Domain Name System (DNS). (Postel, 1980)

2.10 Application layer

The protocols found at the Application layer generally are the protocols that interact with the application running on the host. The Application layer protocol that is covered in more detail in this work is the Simple Network Management Protocol and will be discussed further in the next chapter. Some of the applications that make use of this protocol are listed in the next section — the implemented SNMP software agent also falls under this category.

2.11 Network monitoring software

Several network monitoring software packages for use on conventional networks exist. Software is available in both closed-source commercially sold and open-source varieties. Commercial network monitoring systems include HP Openview (HP, 2010b) and HP Network Node Manager (HP, 2010a). In this section some open source network monitoring systems and options will be discussed further. The goal is to reuse this software to facilitate network monitoring on WSNs.

OpenNMS (OpenNMS, 2010b) was the first enterprise-grade network monitoring system (NMS) developed using the open source model (OpenNMS, 2010a). Presently OpenNMS focuses on data collection, service polling and event and notification management. The software is largely written in Java and various Operating Systems are supported including Linux, Solaris, MAC OS X and Windows. OpenNMS supports data collection using the SNMP. At the time of writing IPv6 was not fully supported. The release of OpenNMS 1.10 in middle 2011 focuses mainly on IPv6.

Zenoss Core (Zenoss Community, 2010) is another open source network management solution that was started by Erik Dahl. Sponsored by Zenoss Inc., Zenoss also offers a commercial product in the form of Zenoss Enterprise. It is available for Linux, Mac OS X, in source code and as a VMWare Appliance. Zenoss does not currently support IPv6.

The Dude (MikroTik, 2010) is a network monitoring application by MikroTik. The software is available for Microsoft Windows, installation instructions for Linux, using the Microsoft Windows emulator Wine, is also available. The Dude is free of charge but not open-source. It does not appear to have support for IPv6.

Nagios (Nagios, 2010) is another popular network monitoring tool. It can be used to monitor applications, services, operating systems and more. Nagios has the ability to send alerts and notifications of network problems. It also has logging and reporting capabilities. Records of outages, alerts and notifications are kept and can be easily viewed. Nagios was used as a monitoring tool and will be discussed further in a later section of this document. Nagios supports Linux, SNMPv1 and IPv6.

Cacti (Cacti, 2010) offers network monitoring with a graphing solution. Cacti is a front end to RRDTool. It is PHP driven and gathered data is stored in a MySQL database. RRDTool (Oetiker, 2009) is a open-source, high performance data logging and graphing system. Cacti can gather and graph data from an assortment of sources including SNMP. Cacti will also be discussed further later in this document. Cacti supports Linux, SNMPv1 and IPv6.

The core functionality of the NMS chosen is that it should support IPv6. A second requirement, that is particular to this work, is that the NMS be able to run on a Linux operating system. It is also important that the NMS support version one of the SNMP, as this is the version that will be implemented in the software agent. A summary of the network monitoring systems can be seen in Table 2.1.

Table 2.1: Network monitoring systems summary

NMS	**Linux**	**SNMP**	**IPv6**
OpenNMS	✓	✓	✗
Zenoss Core	✓	✓	✗
The Dude	✓(Wine)	✓	✗
Nagios	✓	✓	✓
Cacti	✓	✓	✓

The packages Nagios and Cacti, normally used on conventional networks, were selected to be used with the implemented SNMPv1 software agent to facilitate network monitoring on WSNs.

2.12 Summary

This chapter gave a background and overview of the topics and concepts covered in the research. These included an introduction to WSNs, layered communication models in which protocols are organised and the relevant protocols encountered in 6LoWPAN WSNs. Various WSN operating systems, hardware and network monitoring software were also looked at.

In the following chapter the SNMP and 6LoWPAN are discussed along with the different versions and concepts found in the SNMP. The structure of an SNMP GetRequest message will also be shown with the relevant fields and their purpose.

Chapter 3

6LoWPAN and SNMP

3.1 6LoWPAN

The two documents that the 6LoWPAN (IPv6 over Low power WPAN) IETF Working group are responsible for, RFC 4919 edited by Kushalnagar, Montenegro & Schumacher (2007) and RFC 4944 edited by Montenegro, Kushalnagar, Hui & Culler (2007), describe the transmission of IPv6 packets over IEEE 802.15.4 networks and the problems associated with it. RFC documents or Request for Comment documents are the official method or channel of publishing of the Internet Engineering Task Force (IETF). Informational documents and Internet Standards are both published through this channel as RFCs.

The use of an IP-based network stack in WSNs has various advantages. The ubiquity of IP networks allow the use of existing infrastructure and tools when connecting WSNs to other IP networks, for example proxies and firewalls. Connecting IP-based WPANs should be much easier than connecting WPANs using proprietary network protocols. IP technology is well known, fairly mature and the specifications and standards regarding IP networks are open and freely available. Tools and techniques for managing and diagnosing IP networks are already available. IPv6 provides mechanisms for stateless auto configuration, which is useful when dealing with large numbers of nodes. IPv6 can also more than meet the need for address space in WPANs, where deployed devices are envisioned to be in their thousands. (Kushalnagar *et al.*, 2007)

Hui (2008) suggests that the 'narrow-waist' of the protocol stack used in WSNs be placed at the Network layer. The narrow-waist refers to a single common protocol used by different network stacks. This means that varying upper and lower layer protocols can be used without much difficulty because of the common protocol on each stack. This can be done through the use of IPv6 and 6LoWPAN. It makes complex application gateways or proxies unnecessary when connecting IP-based WSN nodes to other IP-based networks.

According to Hui (2008) the introduction of an Active Message Dispatch ID in TinyOS messages rather than more conventional header formats caused the protocols developed by the community to operate at the Link layer rather than the Network layer. The Active message layer provides only minimal services like multiplexed access to the radio transceiver, an Active Message Type field (similar to the Ethernet frame Type field) and single hop message delivery by Active Message address. Other functionality is left to the user to implement. This is in contrast to the traditional IP-based system abstraction. The serial interface to a 'basestation' mote also naturally leads to application level gateways at the edge of the WSN (Hui, 2008).

As stated in the previous section, IPv6 specifies a minimum MTU (Maximum Transmission Unit) of 1280 octets for the link. This is larger than the IEEE 802.15.4 link layer MTU of 127 bytes. This situation is made worse by the addition of Network and Transport layer headers that, in some cases, could leave as little as 33 (Montenegro *et al.*, 2007) or 22 (Hui, 2008) octets for application data, depending on the scenario.

To comply with IPv6 requirements, a fragmentation and reassembly adaptation layer needs to be implemented between the Link and Network layer (a layer below IP). The main function of the adaptation layer is to deal with the problem of the IPv6 minimum MTU and Physical layer packet size for IEEE 802.15.4 networks by providing a fragmentation and reassembly service. Large IP datagrams that do not fit into a single IEEE 802.15.4 frame are divided into smaller fragments that satisfy the MTU for the Link layer. These fragments are then reassembled at their destination (or as specified by IPv6). This adaptation layer forms part of 6LoWPAN.

Header compression can be used to reduce the size of headers before transmission. Header compression also makes the use of IPv6 more practical in WSNs. It is likely that Application layer protocols will produce messages large enough to cause fragmentation when the required headers for lower layers are added. Reducing the size of the headers will make sure that unnecessary fragmentation and reassembly do not occur. It is expected that the most applications of IP over IEEE 802.15.4 will make use of header compression (Montenegro *et al.*, 2007). The RFC dealing with transmission of IPv6 packets over IEEE 802.15.4 networks (Montenegro *et al.*, 2007) specifies a basic method of compressing IPv6 and UDP header fields using stateless header compression. UDP header fields can be compressed using HC_UDP encoding as stated in the RFC, while IPv6 header values can be compressed using HC1. The RFC also summarises some differences between header compression of IPv6 packets for IEEE 802.15.4 links and other standardised header compression schemes. A draft document (Hui & Thubert, 2009) that still has a 'work in progress' status will, if approved, update the compression format found in RFC 4944 adding more effective compression of unique local, global, and multicast IPv6 addresses as well as commonly used IPv6 Hop Limit values. The draft also specifies an encoding format for arbitrary next headers.

3.1.1 6LoWPAN implementations

The initial 6LoWPAN implementation for TinyOS was by Matus Harvan. The 6LoWPAN adaptation layer that was present could handle fragmentation of IPv6 datagrams, mesh addressing and broadcast headers. 6LoWPAN HC1 header compression and HC_UDP UDP header compression was implemented. No multi hop or mesh networking was implemented however. It appears that development and support ceased after initial release. (Harvan, 2007)

Blip (Berkeley IP implementation), formerly *b6LoWPAN*, is the other 6LoWPAN implementation available for TinyOS. This implementation by Stephen Dawson-Haggerty from UC Berkeley supports 6LoWPAN HC1 header compression, point-to-point routing, IPv6 neighbour discovery and default route selection. It supports Epic, TelosB, Iris and MicaZ platforms. The implementation is currently still being actively supported and developed and has been included into the TinyOS core. *Blip* also supports Low Power Listening (LPL) used to duty cycle the use of the radio transceiver to extend battery life. (Dawson-Haggerty & Tavakoli, 2010)

Various commercial 6LoWPAN stacks are available from manufacturers for use with their hardware. Jennic (Jennic, 2010) offers a selection of network protocol stacks for use with their wireless microcontrollers and modules, one of which is a 6LoWPAN stack. Manufacturers like Sensinode (Sensinode, 2010) and Arch Rock (Archrock, 2010) base their software and hardware products on 6LoWPAN technology. An industry alliance around 6LoWPAN and IP based network technology has also been formed called the IPSO Alliance (IP Smart Objects) (IPSO Alliance, 2010). One of the Alliance's aims is to promote and educate users about the use of the IP for connecting Smart Objects.

The *blip* stack was chosen because it offered the needed functionality and extra features like multihop routing and LPL support. It was also developed for use with TinyOS and TelosB, the hardware platform used. The implementation was also still actively being developed and supported at the time of this work. The interfaces to the components were clear and relatively easy to use.

3.2 6LoWPAN and SNMP

The IETF 6LoWPAN Working Group is not officially working on SNMP for LoWPANs, but some 6LoWPAN-related draft documents dealing with SNMP optimisations for 6LoWPAN (Mukhtar, Joo, Schoenwaelder & Kim, 2009) and a 6LoWPAN Management Information Base (Kim, Mukhtar, Joo, Yoo & Park, 2009) has emerged. It is worth noting that in RFC 4919 (Kushalnagar *et al.*, 2007) one of the goals of 6LoWPAN is Network Management and that one of the points of transmitting IPv6 packets is the reuse of existing protocols.

The SNMP is specifically mentioned in RFC 4919 as it is widely used to monitor data sources on conventional networks. The possibility that SNMP functionality may be translated 'as is' to LoWPANs is also noted, as this would enable the reuse of existing tools. It is however stated that SNMPv3 might not be a suitable candidate for use on LoWPANs because of the limitations on resources such as processing, message size and memory.

The documents mentioned above are, at this stage, only Internet-Drafts and works in progress. The SNMP optimisations for 6LoWPAN document states the case for use of SNMP and looks at the third version of the protocol. The documents do not venture far into security for use with the SNMP. Security is one of the big hurdles in terms of available resources on the target devices because of the overhead incurred with the use of the current security and privacy mechanism associated with the SMNPv3. Header compression for the SNMP is also being looked at.

The now expired draft documents do open the floor for discussion around the SNMP and its use with 6LoWPAN, but at this stage do not offer any concrete specifications.

The decision was made to use the first version of the Simple Network Management Protocol for the TinyOS implementation. The first version has none of the security overhead that version three of the protocol has, which could be problematic for the resource constrained devices the agent is intended for. As little in terms of security and 6LoWPAN have been finalised, the issue of SNMP security over 6LoWPAN will not be addressed and falls outside the scope of this document. This would also simplify the implementation and limit the amount of resources needed by the implementation of the agent, leaving resources for the main application running on the WSN. Version one of the SNMP is still well supported by monitoring and management software. The changes or improvements to the subsequent versions of the protocol include, but are not limited to, methods for improved bulk transfer of information and added security. The combination of the small physical layer packet size of IEEE 802.15.4 and the multiple requests possible in a single message when bulk transfer is used would most likely cause frequent fragmentation of the

IP packet, which is not desirable.

Using the *blip* 6LoWPAN implementation it is possible to create an SNMP agent for use on a WSN. This enables the WSN to be monitored and managed using existing software and applications usually used for conventional IP-based networks. There is thus no need to create custom software for this purpose. It is possible to integrate the monitoring of a WSN into existing monitoring infrastructure. It is also possible to use SNMP for the retrieval of sensor data from WSN nodes with no need for other application code on the node.

The implementation of an SNMP agent for the TinyOS WSN operating system demonstrates the basic functionality that using SNMP brings to LoWPANs and also confirm some of the assumed benefits that IPv6 brings. Existing tools are then tested and evaluated against the implementation.

The implementation of an agent that can interpret and produce SNMP messages would also mean that no proxies or gateway entities would be necessary. This has the advantage that no additions or alterations have to be made to the network stack used. Using 6LoWPAN also brings a common protocol to the network that should ease its connection to other networks, including WSNs.

3.3 SNMP

The IAB (Internet Architecture Board) initially recommended, in 1988, both a short- and long-term solution for the development of Internet Network Management Standards. This involved the use of SNMP in the short-term and a more long-term solution in the use of the ISO (International Standards Organisation) network management framework that uses the CMIS/CMIP (Common Management Information Services/Common Management Information Protocol) or CMOT (Common Management Information Protocol on TCP) (Cerf, 1988; 1989). It is worth noting that the predecessor to SNMP is known as the Simple Gateway Monitoring Protocol (SGMP) and is not compatible with the SNMP.

A working group was also established to define the Structure of Management Information (SMI) based on the ISO SMI that would guide the naming and abstraction conventions used in the Internet Management Information Base (MIB). The first versions of the Internet SMI and MIB were specified in RFC 1065 (Rose & McCloghrie, 1988) and RFC 1066 (McCloghrie & Rose, 1988). The SMI and MIB defined in these documents would be common and compatible to both the SNMP and CMOT, facilitating the anticipated move from SNMP to CMOT in the future.

The expected similarities between the SNMP and OSI network management frameworks were, however, not as great as anticipated and the requirement for compatibility between the two frameworks and the SMI/MIB was dropped. This enabled the SNMP to respond to new needs in the Internet community by defining new MIB items. The IAB gave the SNMP, SMI and initial Internet MIB a full 'Standard Protocols' with 'Recommended' status. In doing so the IAB recommended that all IP and TCP implementations become network manageable using the SMI, MIB and SNMP. (Case, Fedor, Schoffstall & Davin, 1990)

In 1990, documents defining the network management framework for IP based networks using the SNMP were published and are now known as SNMPv1. The documents were: '*Structure and identification of management information for TCP/IP-based internets*' (Rose & McCloghrie, 1990); '*Management Information Base for network management of TCP/IP-based internets*' (McCloghrie & Rose, 1990); '*Simple Network Management Protocol (SNMP)*' (Case *et al.*, 1990).

In its most basic form, a network managed using the SNMP consists of three elements namely: the managed device, agent and network management station. The managed device is typically a network node that is connected to the managed network, like a network router or server. On the managed device resides a software agent that gathers information from the managed device that it is running on. The information that is gathered and available is defined in the MIB. The agents present on the managed devices perform network management tasks as requested by network management applications that run on the network management station. A network management station can, for example, request information from the managed device. The agent on the managed device gathers the required information and then responds to the request by sending the appropriate information using the SNMP. (Cisco, 2010b)

The Management Information Base (MIB) is a virtual information store (Rose & McCloghrie, 1990) that contains managed objects that describe the information on the managed device. These managed objects are described using a limited subset of the ASN.1 (Abstract Syntax Notation One) (ITU-T, 2010). Each managed object has a name, syntax and encoding. Each managed object in the MIB is assigned an object identifier, which serves as its name. An object's syntax refers to the abstract data structure to which the object type corresponds. For example, an object might have an object type of integer or octet string. Not all ASN.1 constructs are allowed in the definition of the syntax of a managed object. The constructs that can be used are limited for simplicity's sake. Encoding of the managed object refers to how the object is represented when being transmitted on the network using the object's syntax. ASN.1's Basic Encoding Rules (BER) are used for encoding (ITU-T, 2008).

The first version of the Management Information Base for network management of TCP/IP-based internets was defined in RFC 1156 (McCloghrie & Rose, 1990) and contains a list of managed objects. Later, a second version of the MIB, MIB-2 was defined in RFC 1213 (McCloghrie & Rose, 1991). The mechanisms for describing these objects are defined in RFC 1155: *Structure and Identification of Management Information for TCP/IP-based Internets* (Rose & McCloghrie, 1990). The purpose of this RFC document is not to define managed objects but to specify the format that other RFC documents use to define managed objects.

An object identifier (OID) is a list of numbers that span a global tree from the root to the specific object, similar to an address or path. Each object is assigned a string called an object descriptor for human convenience that can also be used when referring to an object. The tree starts at an unlabeled root node with multiple children or subtrees. Figure 3.1 shows that the unlabeled root node has three children. These are administered by two entities:

- The ITU-T *(International Telecommunication Union - Telecommunication Standardisation Sector* – formerly the *International Telegraph and Telephone Consultative Committee)*, with label ccitt(0) — (object descriptor of *ccitt* and OID of 0),
- the ISO, with the label iso(1) — (object descriptor of *iso* and OID of 1) and
- the third is administered by both the ISO and the ITU-T with the label joint-iso-ccitt(2) — (object descriptor of *joint-iso-ccitt* and OID of 2).

This tree can be expanded and can continue to an arbitrary depth.

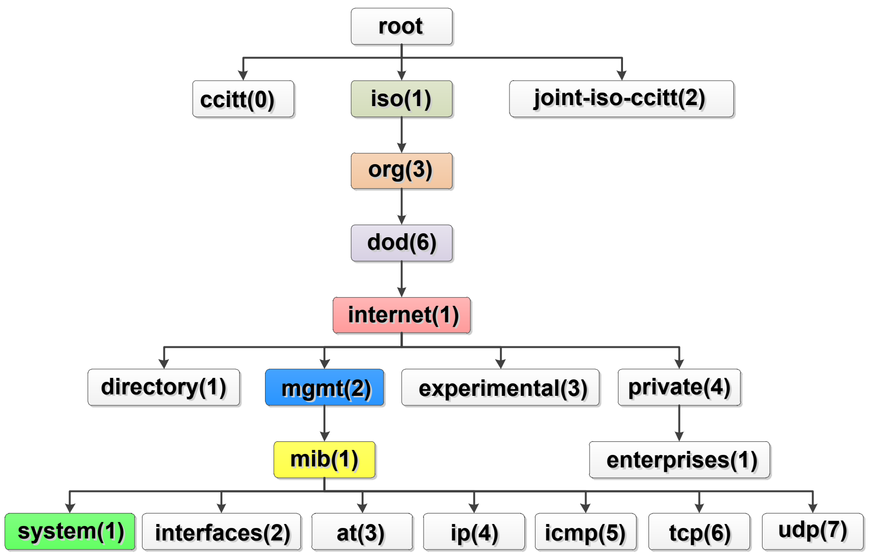


Figure 3.1: The OID tree showing the structure of management information

Below its node (iso(1)) the ISO assigned a subtree org(3) that can be used by national and international organisations. Two of the nodes under this subtree have been assigned to the U.S. National Institute of Standards and Technology. Of those nodes, dod(6), was then transferred to the U.S. Department of Defence. The IAB has reserved a node under the dod(6) subtree for use by the Internet community to be administered by the IAB, internet(1). This means that the Internet subtree can be identified by the object identifier 1.3.6.1 or iso.org.dod.internet. This also forms the start of all identifiers for objects under the Internet subtree. (Rose & McCloghrie, 1990)

RFC 1155 (Rose & McCloghrie, 1990) specifies the policy under which the Internet subtree is managed and initially specifies four nodes under this subtree: directory(1), mgmt(2), experimental(3) and private(4). The first subtree, directory(1), is reserved for future use. The second mgmt(2) subtree contains the object identifiers that identify different versions of the Internet standard Management Information Base (MIB). The initial Internet standard MIB is identified by the number 1. This means that the initial Internet standard MIB can be identified by the object identifier 1.3.6.1.2.1. The experimental(3) subtree contains identifiers for objects used in Internet experiments. The private(4) subtree, that is assigned the identifier 4 and is administered by the Internet Assigned Numbers Authority (IANA), was initially declared with one child; enterprises(1). Manufacturers or enterprises can apply for a subtree under the enterprises(1) subtree. This allows the manufacturers to define their own managed objects or MIBs. It is also recommended that manufacturers register their networking subsystems under their assigned subtree. This means that a manufacturer can request a child under enterprises(1) tree from the IANA, for example 1.3.6.1.4.1.38 . Network subsystems made by the manufacturer can then register under that subtree, for example 1.3.6.1.4.1.38.1.1 .

As an example, the sysDescr object that carries a text description of the managed device can be identified by its object identifier, 1.3.6.1.2.1.1.1.0, or by its object descriptor, iso.org.dod.internet.mgmt.mib.system.sysDescr. The relation between the object identifier, object descriptor and the tree in which it is organised in is indicated by the colour coding in Figures 3.1 and 3.2.

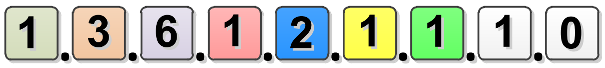


Figure 3.2: Colour-coded object identifier for sysDescr object

3.3.1 SNMPv1

The Simple Network Management Protocol version one (SNMPv1) was the initial version of the SNMP and is specified in RFC 1157 (Case *et al.*, 1990). Together with RFC 1155, which defines the SMI describing how managed objects are defined in the MIB, and RFC 1156, which defines objects present in the MIB, RFC 1157 forms the first incarnation of the SNMP. The SNMP is used to convey management information between network management stations and software agents running on the network elements. The SNMP handles all management as reading or changing of variables. This essentially limits the number of commands used in the SNMP to two, an operation to obtain a value from a managed device (get) and an operation to change a given value on a managed device (set). The management station thus has to poll for the information it requires. A network management agent is also able to send unsolicited messages to a network management station (traps). A group consisting of an arbitrary number of network management stations and SNMP agents is termed an SNMP community. Each community is identified by its given community string. In SNMPv1 this is used as a very rudimentary authentication method.

The SNMP only requires that an unreliable datagram service, RFC 1157 specifies UDP (User Datagram Protocol), be used and that an SNMP message fit into a single transport packet encoded using the Basic Encoding Rules of ASN.1. Each SNMP message comprises of an SNMP version identifier, SNMP community name and protocol data unit (PDU), as shown in Figure 3.3. An SNMP message is received at port 161 and SNMP trap messages are sent to port number 162. The SNMP specifies five PDUs: GetRequest-PDU, GetNextRequest-PDU, GetResponse-PDU, SetRequest-PDU and Trap-PDU.

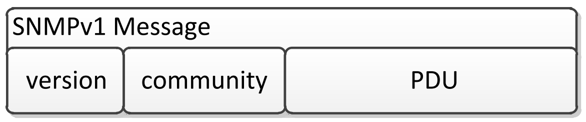


Figure 3.3: SNMPv1 packet structure

The GetRequest-PDU, GetNextRequest-PDU, GetResponse-PDU and SetRequest-PDU all have the same basic structure and use the same common constructs. The Trap-PDU differs from these PDUs, as shown later. Figure 3.4 shows the basic structure of the SNMPv1 protocol data unit (PDU). The RequestID is used to identify individual messages and detect duplicate messages. ErrorStatus and ErrorIndex provide information on errors that might have occurred. ErrorStatus indicates what error occurred while ErrorIndex can give additional information on which variable in the list of variables in the PDU caused the error. When no error is present the value of ErrorStatus is zero. Possible errors and their codes include: tooBig(1), noSuchName(2), badValue(3), readOnly(4) and genErr(5). A VarBindList, or variable binding list, is a list of variable name and variable value pairings. The word variable is used to refer to an instance of a managed object. A VarBind or variable binding is used to refer to a variable's name and its matching value. Some PDUs however only use the variable name, for example GetRequest-PDU. In these cases the value part of the variable binding is ignored. Although the value part of the binding is not used it is recommended that it must still be encoded using the ASN.1 NULL value.

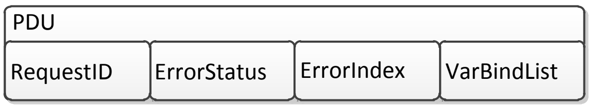


Figure 3.4: SNMPv1 PDU

A GetRequest-PDU is generated when a network management station wants to retrieve information from a managed device. The GetRequest-PDU contains the variable name, in the form of an OID, corresponding to the managed object whose value is to be retrieved. The managed device responds to the request with a GetResponse-PDU that is identical to the GetRequest-PDU with the value of the variable added. A SetRequest-PDU is generated when a management station wants to set or change the value of a managed object on a managed device. The SetRequest-PDU contains the variable name of the managed object as well as the new value of the variable. After receiving the SetRequest-PDU the managed device responds with a GetResponse-PDU that contains the variable name and new value. A managed device responds to a GetNextRequest-PDU with a GetResponse-PDU that contains the variable name and value of the variable immediately after the requested one in the tree. One of the uses of the GetNextRequest-PDU is for traversing conceptual tables containing information on a managed device.

Managed devices are also able to send unsolicited messages to network management stations. This can be likened to a managed device sending a GetResponse-PDU without having received a GetRequest-PDU. This type of message is generated by sending a Trap-PDU. The structure of the Trap-PDU differs from the other PDUs in the SNMPv1 as can be seen in Figure 3.5. SNMP trap messages are sent to UDP port 162. The enterprise field contains the OID of the entity or organisation that is generating the trap, for example 1.3.6.1.4.1.26484. The agent-addr field contains the IPv4 address of the SNMP agent generating the trap message. If an agent does not use IPv4 this field is given the value 0.0.0.0 as pointed out by Thaler (2002). The generic-trap field indicates the type of trap generated, with the specific-trap field used to indicate enterprise specific traps. The time-stamp field contains the current uptime of the agent. The VarbindList is used in the same manner as the previously described PDUs. PDUs used by the SNMPv1 and their respective fields are documented in the RFC edited by Case *et al.* (1990).

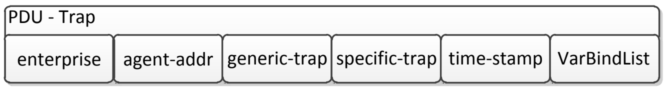


Figure 3.5: SNMPv1 Trap PDU

3.3.2 SNMPv2

In 1993 the second version of the SNMP was defined and built on the first version of the protocol (Case, McCloghrie, Rose & Waldbusser, 1993). A second version of the Structure of Management Information (SMIv2) document, RFC 2578 (McCloghrie, Perkins, Schoenwaelder, Case, McCloghrie, Rose & Waldbusser, 1999b), was also published for use with the SNMPv2. A document describing the Management Information Base for SNMPv2, RFC 3418 (Presuhn, Case, McCloghrie, Rose & Waldbusser, 2002a), was published and made use of the SMIv2. A document that described the protocol operation of the SNMPv2 was also published (Presuhn, Case, McCloghrie, Rose & Waldbusser, 2002c). As with SNMPv1, it is preferred that the UDP be used as a Transport Protocol, but other transport mappings are also possible (Presuhn, Case, McCloghrie, Rose & Waldbusser, 2002b).

The Administrative Model for SNMPv2 (Galvin & McCloghrie, 1993a) describes the behaviour of an SNMPv2 party. Under the SNMPv2 each party uses a single privacy protocol and a single authentication protocol as defined by the Security Protocols for SNMPv2 document (Galvin & McCloghrie, 1993b). These added security features are some of the differences between the SNMP version one and version two.

The Security Protocols for SNMPv2 document defines one authentication protocol that ensures that a message sent from a party can be correctly identified as having come from that party. It also ensures that the message that was received from a party is the same as the message that was sent. This is done with a Digest Authentication Protocol and uses the MD5 message digest algorithm with a 128-bit digest (Galvin & McCloghrie, 1993b).

The Security Protocols for SNMPv2 document also specifies a privacy protocol that ensures that a message sent from one SNMPv2 party to another cannot be read by a third party. Privacy is ensured by using a Symmetric Privacy Protocol that uses the Data Encryption Standard (DES) in the Cipher Block Chaining mode (Galvin & McCloghrie, 1993b).

The Security Protocols for SNMPv2 document (Galvin & McCloghrie, 1993b) is however only present in the first set (RFC 1441 - RFC 1452) of RFCs for SNMPv2. This set is sometimes called SNMPv2p or the Party-based SNMPv2 (Case, Mundy, Partain & Stewart, 2002). The second set of RFCs (RFC 1902 - RFC 1908), some of which obsolete documents from the first set, does not list any documents dealing with security. The IETF SNMPv3 Working Group page states, in terms of improvements to security, that '...strongly held differences on how to incorporate these improvements into SNMP prevented the SNMPv2 Working Group from coming to closure on a single approach. As a result, two different approaches (commonly called V2u and V2*) have emerged' (SNMPv3WG, 2002). Wijnen (1996) presented a comparison between the two approaches.

In contrast to the proposed added security features of the SNMPv2 a memo was published that describes Community-based SNMPv2, also known as SNMPv2c (Case, McCloghrie, Rose & Waldbusser, 1996a). The protocol does away with any added security features implemented in SNMPv2 and uses a community-based administrative framework that is based on SNMPv1. The SNMPv2c does however use SNMPv2's new PDU types and error codes (Presuhn *et al.*, 2002c). This version had the most support within the IETF but lacked security. (Case *et al.*, 2002)

As mentioned above, two other SNMPv2 variants exist, SNMPv2u (McCloghrie, 1996; Waters, 1996) and SNMPv2* (SNMPv3WG, 2002; Case *et al.*, 2002). These versions, in contrast to SNMPv2c, implemented the added security needed but did not have a consensus of support within the IETF (Case *et al.*, 2002).

SNMPv2 offers some improvements over SNMPv1 but some of the functionality from version one remains the same in version two. The Set, Get and GetNext request operations are the same as in version one, but some new operations are added to version two. These include the GetBulk request and Inform operation. The Trap operation from version one is still present in version two, but uses a different message format. SNMPv2 also added expanded data types, for example a 64-bit counter.

The GetBulk operation offers an efficient way to retrieve large blocks of information. It can also be used in a similar fashion to GetNext to traverse tables. When an SNMPv2 Trap message is sent there is no confirmation of receipt. An SNMPv2 Inform message does however offer confirmation of delivery (Presuhn *et al.*, 2002c). This can be seen as a more reliable form of Trap message.

3.3.3 SNMPv3

The third version of the SNMP is derived from, and builds upon, the two previous versions of the protocol. The SNMPv3 Working Group's core focus was to define added security and administration to the SNMP management framework that would enable secure communication of management data. This security includes authentication and privacy mechanisms. The IETF Working Group (WG) for SNMPv3 was tasked to create a single standard and set of documents to provide the needed security, something that the SNMPv2 Working group could not complete. The SNMPv3 WG was chartered to produce a single set of documents based on elements and concepts of SNMPv2u and SNMPv2* (Case *et al.*, 2002).

SNMPv3 continued the use of a modular architecture. All three versions of the Internet Standard Management Framework share the same underlying structure. This enabled the use of previous SNMPv2 draft standards where possible, making it possible for the Working Group to focus on the core task at hand, security, while not 'reinventing the wheel' (Case *et al.*, 2002). The modular design also had the benefit that documents could be upgraded or replaced if necessary without affecting the other documents or modules.

SNMPv3 also uses the Structure of Management Information Version 2 (SMIv2) (McCloghrie *et al.*, 1999b; McCloghrie, Perkins, Schoenwaelder, Case, McCloghrie, Rose & Waldbusser, 1999c;a) from SNMPv2 as a data definition language for describing MIB modules.

Specifications for the operation of SNMPv3 can be found in RFCs 3410 to 3418. These documents include a Protocol Operations RFC (Presuhn *et al.*, 2002c) that is based on and updates the document for Protocol Operations for the SNMPv2 (Case, McCloghrie, Rose & Waldbusser, 1996b), thus building on and reusing previous Draft Standards. The Transport Mappings RFC from SNMPv2 is also reused in an updated form, RFC 3417 (Presuhn *et al.*, 2002b). RFC 2576 (Frye, Levi, Routhier & Wijnen, 2000) describes the coexistence between SNMPv3, SNMPv2 and SNMPv1.

RFC 3414 (Blumenthal & Wijnen, 2002) and RFC 3415 (Wijnen, Presuhn & McCloghrie, 2002), individually describe a User-Based Security Model (USM) and View-based Access Control Model (VCAM) for the SNMPv3. USM provides SNMP message level security while VCAM controls access to management information. RFC 3414 describes using HMAC-MD5-96 (MD5 hash-function (Rivest, 1992) in Hash-based Message Authentication Code (HMAC) mode (Krawczyk, Bellare & Canetti, 1997)) and HMAC-SHA-96 (SHA hash-function (NIST, 1995) in HMAC mode (Krawczyk *et al.*, 1997)) as authentication protocols. The memo also describes the use of the CBC-DES (Data Encryption Standard (DES) in the Cipher Block Chaining mode) Symmetric Encryption Protocol as a privacy protocol. The memo notes that it is possible to replace or supplement these protocols in the future.

3.4 Anatomy of an SNMP message

Although five different types of SNMPv1 messages exist (GetRequest, GetNextRequest, GetResponse, SetRequest and Trap), it is possible to look at one of the types and get a general understanding as to how an SNMP message is structured. In this section an SNMPv1 GetRequest message will be investigated and the different parts of the message examined.

The diagram in Figure 3.6 shows an SNMP GetRequest message as generated by a NMS (Network Monitoring Station). To ensure that common data types and encoding of values are used across implementations in different languages, ASN.1 and its Basic Encoding Rules (BER) are used in the SNMP. SNMP messages are generally received at UDP port 161 of the network node being monitored.

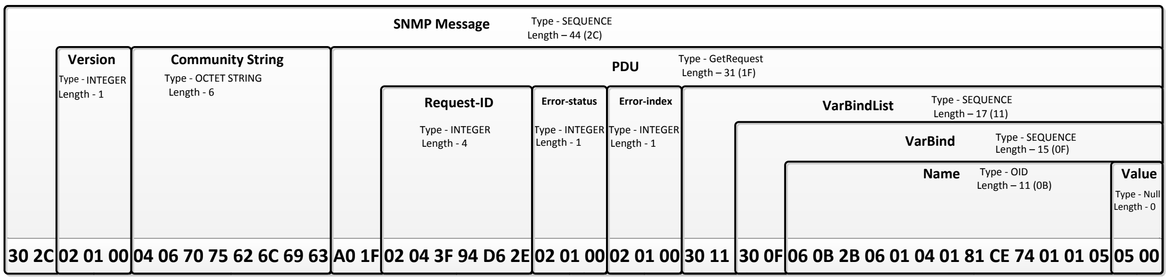


Figure 3.6: SNMP GetRequest message with values shown

Each field in the message consists of three parts. The first part indicates the **Type** of the data contained in that field. The second part indicates the **Length** of the next part of the message that contains the data (Value). The third and last part contains the data or **Value**, which is of the type that part one indicates and of length that the second part specifies. This is as per the *type*, *length*, *value* (TLV) encoding of the Basic Encoding Rules (BER). Figure 3.7 shows a diagram of the encoding and an example of the encoding of the SNMP version number from the GetRequest message shown in Figure 3.6. The *02* shows that the **Value** is an integer, as per Table 3.1. The *01* shows that the **Value** field is one octet long, with the *00* indicating version one of the SNMP.

Type	Length	Value
02	01	00

Figure 3.7: Type Length Value encoding of BER

From left to right the first two values (`30 2C`) in Figure 3.6 shows the Type (`30`) and Length (2C) in octets of the **SNMP message** with the last field containing the rest of the message as a Value field (with all values in hexadecimal) as per the TLV encoding. The next part of the message contains the **Version** number of the SNMP being used. This is followed by the **Community String**. The next part contains the Protocol Data Unit (**PDU**) of which five are possible: GetRequest-PDU, GetNextRequest-PDU, GetResponse-PDU, SetRequest-PDU and Trap-PDU. The corresponding Identifier for some commonly used data types are shown in Table 3.1.

Table 3.1: A short list of ASN.1 data types used with the BER

Data type	**Identifier**
GetRequest	0xA0
GetNextRequest	0xA1
GetResponse	0xA2
SetRequest	0xA3
Trap	0xA4
Integer	0x02
Octet string	0x04
Sequence	0x30
Null	0x05
OID	0x06

The Value field of the PDU contains the rest of the SNMP message. The **Request-ID** field contains an identifier unique to that specific message and is used to detect duplicate messages and to compare outstanding messages with incoming messages.

The **Error-status** and **Error-index** fields indicate the error status of the message, which is typically not used in GetRequest messages. Possible values for the Error-status field include: noError(0), tooBig(1), noSuchName(2), badValue(3), readOnly(4) and genErr(5). It is possible for the variable bindings list (VarBindList) field to contain more than one variable binding (VarBind). If this is the case the Error-index field points to the VarBind that caused the error.

Some examples of error messages: if a GetRequest is sent to a node and the object name does not exactly match any identifier in the node's MIB, then the node responds with a GetResponse message that is identical to the received GetRequest message but the Error-status field is set to noSuchName(2) and the Error-index field points to the VarBind involved (Case *et al.*, 1990). If a GetResponse message is generated (after receiving a request message) that exceeds a size limit on the node, the node replies with a GetResponse message identical to the request but with Error-status set to tooBig(1) and Error-index to a zero value. If the contents of the value field in the VarBind does not match the expected type, length or value, a response is generated with the Error-status field set to badValue(3). If an error is generated that is not specified by any of the identifiers, then a genErr(5) is signalled and if no error is present the Error-status field contains the noError(0) identifier.

The **VarBindList** field contains a list of variable bindings, as described in section 3.3.1. The SNMPv1 implementation in this work only makes provision for a single VarBind in the list. This was done for simplicity and due to the constraints imposed by working on a WSN platform. The **VarBind** field contains the **Name** of the variable being requested from the managed node and also a **Value** field (the Name and Value fields make up the data part of the VarBind field). Some PDUs are only concerned with the Name of the variable and not its Value and the PDU of the GetRequest message is one of these. In such a case the SNMP agent on the managed node ignores the Value field, but even though the Value field is ignored it must still have a valid ASN.1 syntax and encoding. It has been recommended that the ASN.1 NULL value be used for the Value field of the variable binding. (Case *et al.*, 1990)

The Value part of the **Name** field in the variable binding in Figure 3.6 is encoded in a different format from the other Value fields in the SNMP message. The Type field contains the identifier for the OID type (`06`) with the next field indicating the length (`0B`). The third field (Value field) contains the OID and is encoded by taking the first value in the OID multiplying it with 40 and adding the second value, $1_{10} * 40_{10} + 3_{10} = 43_{10} = 0x2B$. In this way the first part of the OID `1.3.6.1.4.1` is encoded as `2B 06 01 04 01`, which reduces the encoded length by one octet. This encoding of an object identifier value is done as described by the ITU-T (2008) using ASN.1's Basic Encoding Rules (BER).

3.5 Summary

This chapter gave an overview of 6LoWPAN, some implementations of the specification and the three versions of the Simple Network Management Protocol. It also looked at the current relationship between 6LoWPAN and SNMP. In section 3.4 the structure of an SNMP GetRequest message was shown along with the different fields and their purposes.

The following chapter covers the implementation of the SNMPv1 agent and shows relevant code snippets to aid explanation.

Chapter 4

SNMP agent implementation

In this chapter, the implementation of the SNMP software agent in TinyOS will be discussed. The manner in which the agent implementation is divided and organised will be shown. The different tasks for which each logical block is responsible will also be explained.

As stated, the 6LoWPAN implementation used is the *blip* stack developed at the University of California. The stack provides a means of sending IP datagrams over a WSN. *blip* currently offers UDP and an experimental TCP transport protocol. As the SNMP uses UDP, *blip* can be used to facilitate communication over a WSN using the SNMP.

The *blip* implementation is written for TinyOS, which is an event based operating system designed to be used on wireless sensor network devices. Applications in TinyOS are written using nesC, an extension of the C programming language. A nesC application is divided into components that are wired together to form functional applications. The functionality that a component provides to other components, or uses from another component, is described through interfaces. An interface may have commands or events associated with it that should be implemented or handled by the components involved.

Two types of components exist, modules and configurations. Modules provide the implementation of the interfaces, while configurations are used to wire various components together. Interfaces provided by one component are wired to interfaces of another component that uses them. For example, the functions or tasks that a TinyOS application has to perform can be divided up and then implemented in different modules. These modules can then be connected or wired together using a configuration file. These modules can then be reused and combined with other modules using a new configuration to form a new application.

4.1 SNMP software agent overview

The SNMP agent is divided into two separate logical blocks, as can be seen in Figure 4.1. The first is the `snmp` block and consists of the `snmpC` configuration and `snmpP` module file. The second block is `snmpMib`, this block is divided into the `snmpMibC` configuration and `snmpMibP` module.

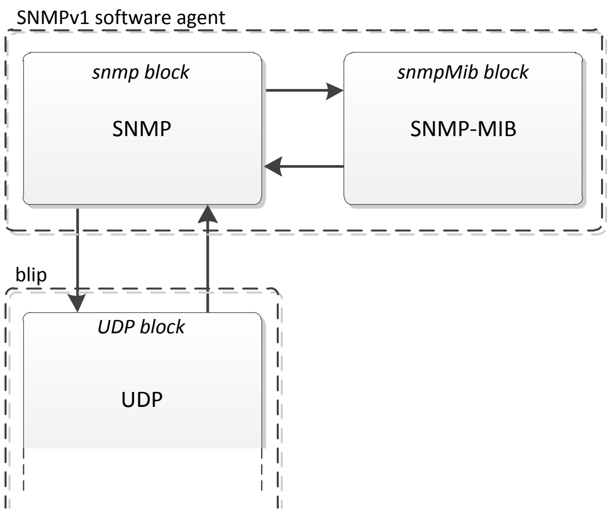


Figure 4.1: SNMPv1 software agent general flow diagram

The `snmp` block is responsible for receiving the raw SNMP message through the UDP interface provided by the UdpSocketC component in *blip* (this is referred to as the UDP block in Figure 4.1). The `snmp` block listens on port number 161 for incoming messages, it then interprets the messages and extracts the needed information. The message is checked to ensure that it is an SNMPv1 message and the SNMP community string is determined. The type of SNMP message and PDU variable name is then extracted. If an SNMP PDU variable value is present, as for a *SetRequest*, it is also extracted. The information that is extracted from the SNMP message is made available through various interfaces provided by the component. The `snmp` block also provides interfaces that can be used to send response messages. The `snmp` block is responsible for constructing and sending SNMP Trap messages. The functionality for sending Trap messages is also provided via a corresponding interface.

The `snmpMib` block is where the Management Information Base (MIB) segment is implemented. This block is used to further process the requests received by the `snmp` block. The `snmpMib` block interprets the type of PDU, variable name and value and acts accordingly. It is also responsible for the initiation of the sending of Trap messages. The block is responsible for gathering and storing the needed information specified in the MIB. When a request for information is received the `snmpMib` block handles the request, gathers the requested information and sends a reply to the request. This happens via the `snmp` block. The `snmpMib` block also provides interfaces for use by the WSN application that it is wired to.

This division of the SNMP agent into two logical blocks with separate tasks is deliberate. This enables the easy creation of many different MIB modules. MIBs with different information can be created and easily wired to the `snmp` block.

Various standard Internet MIBs exist (McCloghrie & Rose, 1990; 1991) and it is expected that SNMP implementations should support these MIBs. It is also possible for companies to define their own Enterprise MIBs in order to better support their specific equipment, these MIBs are housed under the *private(4)* and *enterprise(1)* branches. It was decided not to support the standard Internet MIBs because they contain information and objects that might not be relevant in LoWPANs. The main reason, however, for the omission of the standard MIBs is the limitations in terms of resources on the WSN nodes. A temporary MIB was created and housed under the *enterprise* branch for testing.

4.2 The snmp block

As described earlier, the SNMPv1 software agent is divided into different distinct logical blocks. The `snmp` block consists of the `snmpC.nc` configuration file and the `snmpP.nc` module. This block receives the raw SNMP message from the *blip* 6LoWPAN stack and processes it to extract the needed information from the message. It is also responsible for constructing the SNMP response message that is sent back to *blip* and on to the NMS. The `snmp` block also constructs the SNMP Trap messages that is sent by the node.

Figure 4.2 shows a wiring diagram indicating the other components wired to the `snmpP` component and the interfaces used, while Figure 4.3 shows a generalised flow diagram for the component.

A wiring diagram shows the connections wired between the different components that make up a specific TinyOS application. It shows the interfaces provided and used by components. The rectangular blocks represent the components in the diagram, while the oval blocks indicate provided interfaces. The arrows point towards the component providing the interface, with the interface involved indicated on the line. If a component provides more than one instance of the same interface, the different instances of the interface can be renamed using the `as` keyword. This can also be done for clarity's sake. An example can be seen in Figure 4.2 where the *Statistics* interface was renamed and used as *SNMPStats*. Another example is the *SnmpNotify* interface in Figure 4.6 that is used as *ReportPeriod1*, *ReportPeriod2*, *ReportPeriod3* and *LEDSenable*. This wiring of components is described by the nesC configuration files.

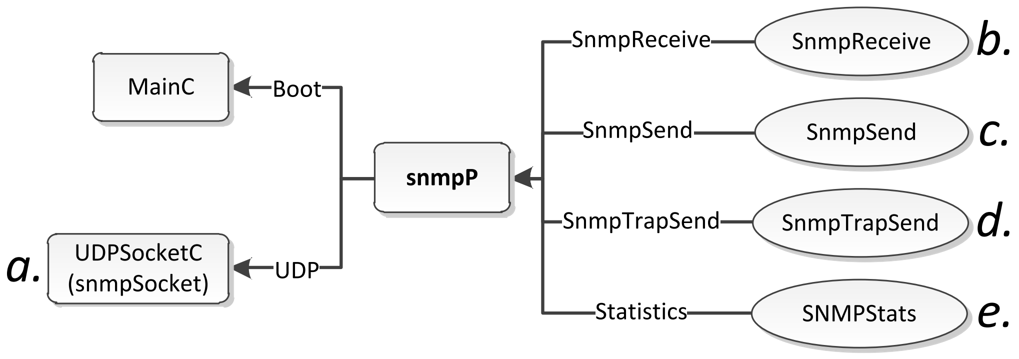


Figure 4.2: Wiring diagram of snmpP component

The `UDP` interface provided by the `UdpSocketC` component (Fig. 4.2(*a.*)) is used by the `snmpP` component, as `snmpSocket`, to send and receive UDP datagrams. This socket is not only used to send and receive SNMP get and set messages it is also used to send SNMP Trap messages. When a UDP datagram is received by the node on port 161, to which the socket is bound, a `recvfrom` (receive from) event is generated for the socket (Fig. 4.3(*a.*)):

```
event void snmpSocket.recvfrom(struct sockaddr_in6 *from, void *data1, uint16_t len,
                               struct ip_metadata *meta)
```

Where *from* contains the address of the node sending the message, *data1* contains the raw SNMP message and *len* the length of the message. The *meta* variable is not used in the implementation.

In the `recvfrom` event handler the received message is checked to see if the first value corresponds to the SNMP Sequence identifier (Fig. 4.3(*.b*)) and it is also checked that the SNMP version number is the expected value.

The SNMP community string received is then verified against the SNMP community string that the node belongs to (Fig. 4.3(*.c*)).

If the community string is correct, the statistics kept for the number of SNMP messages received (`stats.rx`) is incremented, the lengths (`calcLengths`) and start and stop indices (`calcStartStopInd`) of the different fields within the message are calculated. The response message is then partially constructed with values that are known and can be assigned at this stage in the handling of the request (Fig. 4.3(*.d*)). This is done in the `structStuffer` (structure stuffer) function with values that are typically present in both request and response messages. A response message is first constructed in a structure called *snmp_message* with an instance *local*. It is then later copied to a buffer before

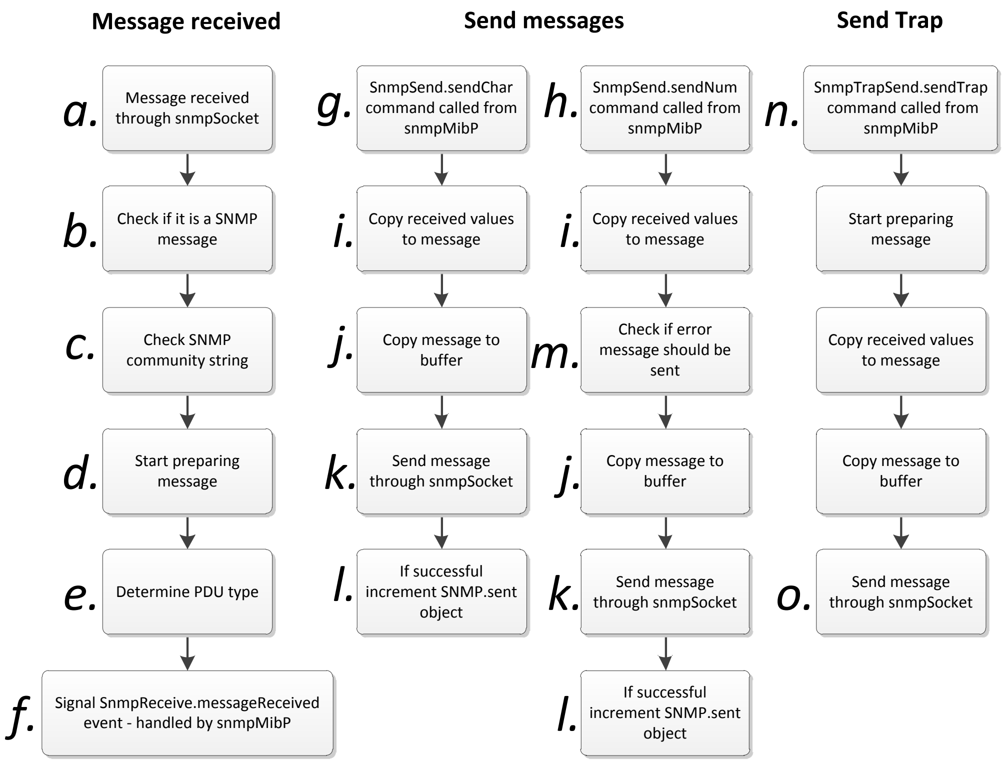

Message received
Send messages
Send Trap
a. Message received through snmpSocket
b. Check if it is a SNMP message
c. Check SNMP community string
d. Start preparing message
e. Determine PDU type
f. Signal SnmpReceive.messageReceived event - handled by snmpMibP
g. SnmpSend.sendChar command called from snmpMibP
i. Copy received values to message
j. Copy message to buffer
k. Send message through snmpSocket
l. If successful increment SNMP.sent object
h. SnmpSend.sendNum command called from snmpMibP
i. Copy received values to message
m. Check if error message should be sent
j. Copy message to buffer
k. Send message through snmpSocket
l. If successful increment SNMP.sent object
n. SnmpTrapSend.sendTrap command called from snmpMibP
Start preparing message
Copy received values to message
Copy message to buffer
o. Send message through snmpSocket

Figure 4.3: General flow diagram of snmpP component

being sent.

The type of PDU received is checked and extracted (Fig. 4.3(*e.*)) along with the OID and object value, if a SetRequest was received. The WSN node is expected to receive only two kinds of SNMP messages, GetRequest and SetRequest. The node only handles receiving these two types and will ignore any other messages. A `messageReceived` event is then signalled through the `SnmpReceive` interface to indicate to the `snmpMibP` module that an SNMP message was received (Fig. 4.3(*f.*)). This event is handled in the `snmpMibP` component or whichever component is wired to the `SnmpReceive` interface (Fig. 4.2(*b.*)).

After the message is processed by the component wired to the `SnmpReceive` interface (Fig. 4.2(*b.*)), in this case `snmpMibP`, the response information compiled by `snmpMibP` is sent back to the `snmpP` component using the `SnmpSend` interface (Fig. 4.2(*c.*)) provided by `snmpP`. For simplicity's sake two commands are provided by the `SnmpSend` interface for sending response messages, one for sending of values contained in a character array (`sendChar`, Fig. 4.3(*g.*)) and a second for numerical values (`sendNum`, Fig. 4.3(*h.*)). These two interfaces could be combined and values handled according to their type identifier. Arguments to the commands are the datatype and value of the requested object and any error messages.

```
command error_t SnmpSend.sendChar (uint8_t SNMPdataType, char *charData, uint8_t SNMPerrror)
command error_t SnmpSend.sendNum (uint8_t SNMPdataType, uint32_t val, uint8_t SNMPerrror)
```

The `sendNum` command is also used when the `snmpMib` block has to respond with an error and the Error-status and Error-index fields have to be set in the SNMP response message (Fig. 4.3(*m.*)).

The handlers for both commands copy the response information into a local structure (Fig. 4.3(*i.*)) for storage before being copied to a buffer for transmission. A function, `copyStructToBuffer` (copy structure to buffer), is then called to copy the local storage structure to a buffer (Fig. 4.3(*j.*)). This buffer is then passed as an argument when calling the `sendto` command from the `UDP` interface provided by `UDPSocketC` and used as `snmpSocket` (Fig. 4.2(*a.*)). This command is used to send the constructed SNMP message to the *blip* stack and on to the network (Fig. 4.3(*k.*)). The number of transmitted messages in the statistics for SNMP messages (`stats.tx`) is then incremented (Fig. 4.3(*l.*)).

The `snmpP` component also offers the interface `SnmpTrapSend` (Fig. 4.2(*d.*)) with the command `sendTrap` for the sending of SNMP trap messages (Fig. 4.3(*n.*)).

```
command error_t SnmpTrapSend.sendTrap (uint8_t genericTrapValue,
                                       uint8_t specificTrapValue, uint32_t timestamp,
                                       char *PDUobjectIDValue , uint8_t PDUobjectType,
                                       void *PDUobjectValue, uint8_t needToConvert)
```

Where *genericTrapValue*, *specificTrapValue* and *timestamp* are the fields shown in Figure 3.5 (repeated here for convience as Figure 4.4) and described in the text. *PDUobjectIDValue* contains the OID that is sent as part of VarBindList in the message. *PDUobjectType* and *PDUobjectValue* indicates the type and value of sent in the VarBindList. *needToConvert* indicates to the handler function if the value needs to be converted to an 8-bit array before being sent.

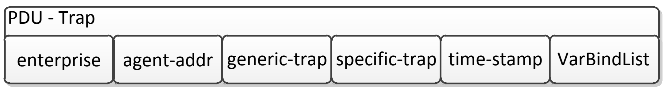


Figure 4.4: SNMPv1 Trap PDU

The SNMP Trap message is prepared and then sent (Fig. 4.3(*o.*)) using the `sendto` command in the `UDP` interface (Fig. 4.2(*a.*)). The `UDP` interface is renamed, and used as `snmpSocket`. The same instance of `snmpSocket` is used to send Trap messages, GetResponse messages and for receiving of Get- and SetRequests. The call to the `sendto` command can be seen below.

```
call snmpSocket.sendto(&snmptrap_address, &send_trap_msg, send_trap_msg_cnt);
```

As mentioned in the previous section the agent implementation only supports one VarBind in the VarBindList. This was done to simplify the implementation and keeping in mind the limited resources available on WSN platforms, while leaving resources for application code to run on the node alongside the SNMP agent. For similar reasons the implementation of the GetNextRequest PDU was also omitted. It was expected that the node would only receive SNMP requests and not generate them. For these reasons the implementation responds to SNMP GetRequest, SetRequests and is able to generate Trap messages and respond with GetResponse messages.

4.3 The snmpMib block

The `snmpMib` block in the SNMP software agent is the block in which the MIB of the agent is implemented, see Figure 4.1. The `snmpMib` block consists of the `snmpMibC.nc` configuration file and the `snmpMibP.nc` module. The `snmpMib` block receives the requested PDU extracted from the raw SNMP message by the `snmp` block. It gathers the requested information from the node and sends it back to the `snmp` block, which generates and sends an SNMP response message with the information. The `snmpMib` block also controls the generation of Trap messages, with the `snmp` block generating the raw SNMP Trap message.

Figure 4.5 shows a generalised flow diagram for the `snmpMibP` component, with Figure 4.6 showing a wiring diagram for the component, which indicates the other components to which it is wired and the interfaces used.

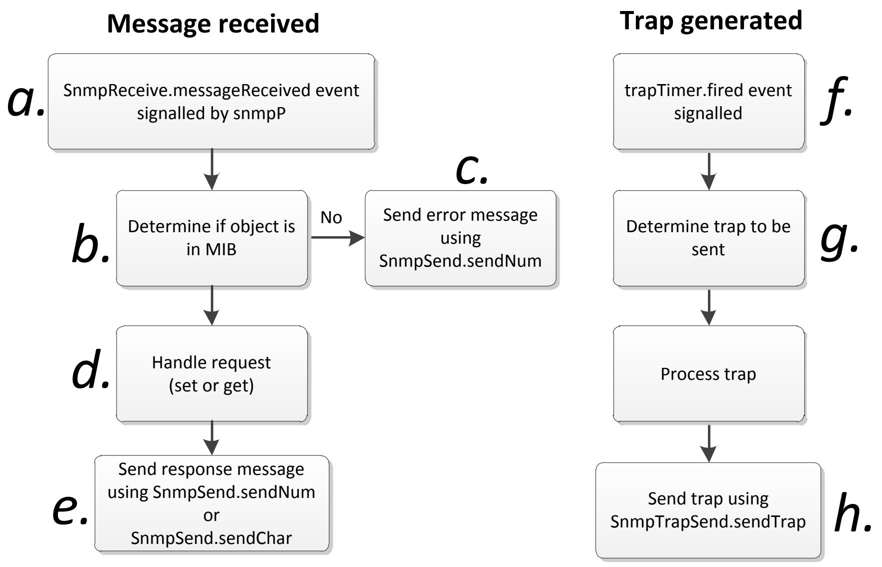

Message received
a.
SnmpReceive.messageReceived event signalled by snmpP
b.
Determine if object is in MIB
No
c.
Send error message using SnmpSend.sendNum
d.
Handle request (set or get)
e.
Send response message using SnmpSend.sendNum or SnmpSend.sendChar
Trap generated
f.
trapTimer.fired event signalled
g.
Determine trap to be sent
Process trap
h.
Send trap using SnmpTrapSend.sendTrap

Figure 4.5: General flow diagram of snmpMibP component

When an SNMP message with a valid SNMP community string is received by the `snmpP` component, a `messageReceived` event is signalled (Fig. 4.5($a.$)) through the `SnmpReceive` interface (Fig. 4.6($a.$)). The arguments to the event include the type of PDU received (GetRequest or SetRequest), OID, length of the OID, datatype of VarBind value received, VarBind value field and length of the VarBind value.

```
event void SnmpReceive.messageReceived (uint8_t PDUtypeReceived,
                                        char *objectIdentifier, uint8_t objectIdentifierLen,
                                        uint8_t objectIdentifierPayloadType, char *objectIdentifierPayload,
                                        uint8_t objectIdentifierPayloadLen)
```

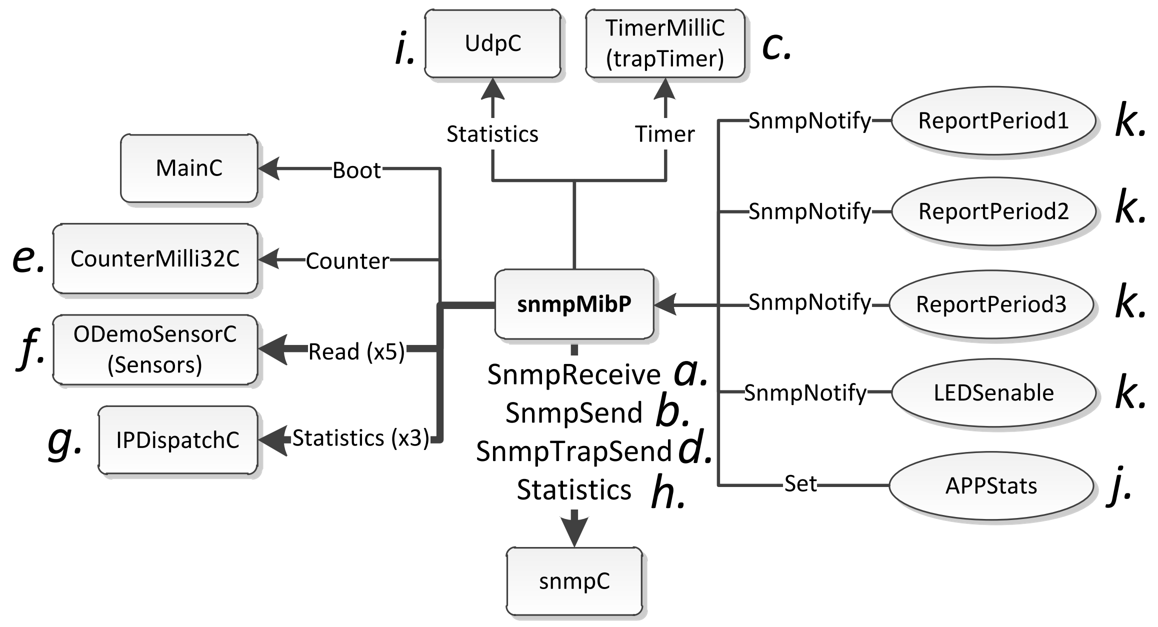


Figure 4.6: Wiring diagram of snmpMibP component

The OID identifying the MIB is defined as `obj_BASE` or the base OID. Each object in the MIB is then defined thereafter. This reduces duplication when defining the OID of an object in the `snmpMib` component. The OIDs are defined as seen below:

```
char obj_BASE[] = {0x2b,0x6,0x1,0x4,0x1,0x81,0xce,0x74,0x1,'\0'};
char OBJ_STATUS = 0x1;
  char obj_sysDescr = 0x1;
  char obj_version = 0x2;
...
```

The above is done instead of defining the full OID for each object, which would be done in the following manner:

```
char OBJ_STATUS[] =   {0x2b,0x6,0x1,0x4,0x1,0x81,0xce,0x74,0x1,0x1};
char obj_sysDescr[] = {0x2b,0x6,0x1,0x4,0x1,0x81,0xce,0x74,0x1,0x1,0x1};
char obj_version[] =  {0x2b,0x6,0x1,0x4,0x1,0x81,0xce,0x74,0x1,0x1,0x2};
...
```

The incoming OID is tested to verify that it is from the MIB maintained by the component (Fig. 4.5(*b.*)). If it is not, the component responds with a noSuchName error message (Fig. 4.5(*c.*)). If the OID corresponds to the MIB implemented it is checked against the IDs of the objects maintained in the MIB. If the object is not in the MIB the component responds with a noSuchName message, but if the object is present the request is processed.

After the requested information is retrieved from the node (Fig. 4.5(*d.*)) a response is sent back to the `snmpP` component (Fig. 4.5(*e.*)) containing the requested information using the `SnmpSend` interface (Fig. 4.6(*b.*)). If a SetRequest message was received, the new value is set in the MIB and a response with the new value is generated and sent back to the `snmpP` component. After a response with the needed information is sent back to the `snmpP` component, the `snmpP` component is responsible for generating the SNMP GetResponse message with the information to be sent back to the querying NMS.

The `snmpMibP` component generates Trap messages with the firing of a timer (Fig. 4.5(*f.*)) called *trapTimer* (Fig. 4.6(*c.*)). The handler for the timer fired event evaluates the *trapState* variable, this variable controls the behaviour of Trap message generation and what Trap message is generated (Fig. 4.5(*g.*)). The frequency at which the timer fires is controlled by the *trapReportPeriod* variable, both *trapState* and *trapReportPeriod* variables are set to a default value in software but can also be changed with an SNMP SetRequest message. In this way the generation of SNMP Trap messages can be controlled remotely by using the SNMP. For testing purposes only two traps were implemented. The valid values that *trapState* can be assigned are listed in Table 4.1.

Table 4.1: Valid values for trapState

Enum	**Value**	**Action**
TRAPSTATE_OFF	0	Traps are off
TRAPSTATE_TEST	1	Send test trap
TRAPSTATE_VOLTAGE	2	Send voltage value

The `sendTrap` command in the `SnmpTrapSend` interface (Fig. 4.6(*d.*)) is then used to send the information gathered for the Trap message to the `snmpP` component to be sent over the WSN to the NMS (Fig. 4.5(*h.*)).

It is possible to set up multiple Trap messages in software and then select what Trap messages to send using SNMP SetRequest messages.

The `Counter` interface provided by `CounterMilli32C` (Fig. 4.6(*e.*)) is used by `snmpMibP` as `Uptime`. This is used to keep track of the time the node has been running and is based on a similar section in the *UDPEcho* application included in *blip*. The `snmpMibP` component also uses a `Read` interface to obtain various sensor readings from the `ODemoSensorC` component (Fig. 4.6(*f.*)). These onboard sensor values are available in the MIB.

The `Statistics` interfaces provided by the `IPDispatchC` component (Fig. 4.6(*g.*)) are used to retrieve information on the IP and ICMP messages sent and received by the *blip* stack and also to gather information from the routing element in *blip*. Another `Statistics` interface provided by the `snmpP` component (Fig. 4.6(*h.*)) provides information on the SNMP messages sent and received by the `snmpP` component. The `UdpC` component also provides a `Statistics` interface through which information is made available about the sent and received UDP messages (Fig. 4.6(*i.*)). This statistics information can be retrieved from the node's MIB using the SNMP.

The `snmpMibP` component provides a `Set` interface (Fig. 4.6(*j.*)) through which the WSN application code running on the node (for example `windStation`, discussed later) is able to indicate to the `snmpMibP` component how many messages it has sent or received. The `Set` interface is used to set the number of sent or received messages in the corresponding object in the MIB. This is used to keep statistics on the number of messages sent and received by the WSN application.

The `snmpMibP` component also provides four `SnmpNotify` interfaces to be used by the WSN application code (Fig. 4.6(*k.*)). These interfaces are used to notify the WSN application code running on the node that the value of the corresponding variable kept in the node's MIB has changed. This can be used to control timers or disable or enable the LEDs on the node through the SNMP.

4.3.1 Management Information Base

The MIB part of the SNMP implementation is housed in the `snmpMibP` component and contains all the managed objects that describe information kept on the managed node. This section describes what happens after the OID has been identified as being in the MIB and what objects are maintained in `snmpMibP`. The MIB description file that makes it possible to translate numeric OIDs to textual object identifiers will also be introduced.

The OIDs are defined with the convention that the part of the OID that identifies the MIB module is defined as `obj_BASE[]` and the object in the MIB is then identified only by its ID within the MIB. This means that for the `sysDescr` only `0x1` has to defined and not the whole `0x2b,0x6,0x1,0x4,0x1,0x81,0xce,0x74,0x1,0x1,0x1` OID (or 1.3.6.1.4.1.26484.1.1.1). This concept is also described in the previous section.

The MIB is currently divided into five sections dealing with status information on the node, onboard sensors of the TelosB platform, variables for external control, route information and statistics on the node. Objects can easily be added to the MIB and also removed to tailor the information available to specific applications. It is also possible to construct new MIBs, which can be selected and wired in the configuration file for specific applications. A diagram depicting the structure of the MIB described in this section can be seen in Figure 4.7

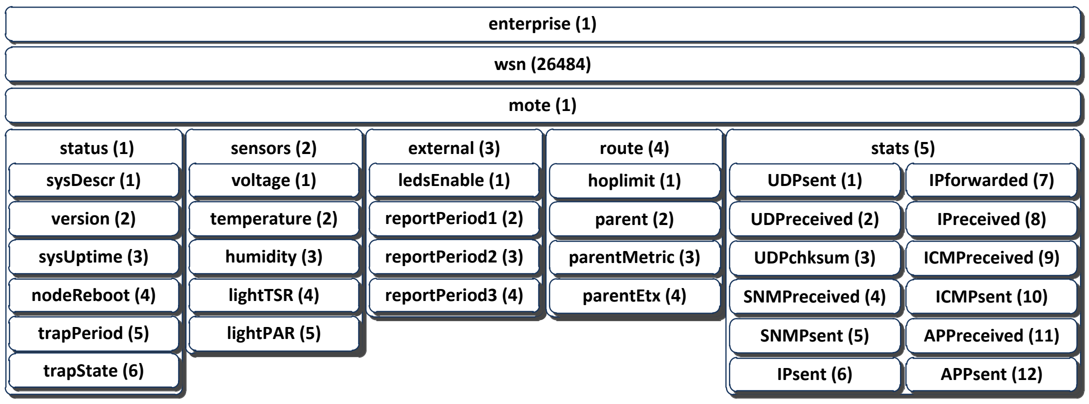


Figure 4.7: Diagram of the WSN-MIB

For testing purposes a temporary identifier of 26484 with an object descriptor *wsn*, under the enterprise(1) subtree, was chosen. Under this subtree a mote(1) child was created that houses the MIB implemented in the `snmpMibP` component.

The first section of the MIB (`wsn.mote.status`) is loosely used for status information on the node. It contains the following objects: `sysDescr`, `version`, `sysUptime`, `nodeReboot`, `trapPeriod` and `trapState`. The first object is a textual description of the node/system and its function. A textual field for identifying the version number of the software running on the node can be found in the `version` object. The `sysUptime` object contains the node's uptime in milliseconds, the uptime however is only incremented in minimum steps of one second. `nodeReboot` is used with SNMP SetRequest messages to reboot the node, if a value of 0xFF is written to the object the node is reset. The node replies with a GetResponse message before it reboots. Both `trapPeriod` and `trapState` are used to control the behaviour of the node's SNMP Trap message generation. `trapPeriod` sets the time between trap messages while `trapState` controls what trap message is sent, if any. For demonstration purposes two traps were defined, the first a test trap that sends a string to the NMS and a second that sends the node's battery voltage. By setting `trapState` to a value of zero, trap generation is turned off.

The next section, `wsn.mote.sensors`, makes the readings from the TelosB platform's sensors available via the SNMP. The onboard sensor levels can be monitored and logged using SNMP GetRequest messages, this also includes the node's battery voltage. The TelosB platform is equipped with the following sensors: temperature, humidity, light - visible, light - visible to IR and supply voltage.

The `wsn.mote.external` section contains variables that can be changed using SNMP SetRequest messages. These variables can be accessed from outside of the `snmpMibP` component by using the `SnmpNotify` interfaces provided by the `snmpMibP` component. These variables can be used to control functions in the main application code running on the node. When an SNMP SetRequest is received and one of the objects is changed an event is signalled that is handled in the WSN application code. An object `ledsEnable` is available to control the LEDs on the node. This can be used to turn LEDs on for debugging and switch them off again for normal operation, using the SNMP. There are also three `reportPeriod` objects that can be used to control timers in the main application code.

Route statistics maintained in the `IPDispatchC` component of the *blip* stack can be accessed using the objects from the `wsn.mote.route` section of the MIB. The `wsn.mote.stats` section provides statistics on traffic to and from the node. Statistics will be discussed further in the next section.

For each object, a handler function is created that is called when the received OID is matched to an object in the MIB. The function performs the necessary actions or gathers the required information and then replies with an appropriate SNMP GetResponse message through the `snmp` component. The example below shows the function for retrieving the number of UDP messages sent by the node. The statistics for UDP messages are retrieved and the number of sent UDP messages is sent to the `snmpMib` block using the `SnmpSend` interface.

```
void handel_obj_MIB_statsUDP_sent ()
{
  udp_statistics_t statsUdp;
  call UDPStats.get(&statsUdp);
  call SnmpSend.sendNum (SNMP_TYPE_UINT32, statsUdp.sent, SNMP_ERROR_noError);
}
```

The MIB description file `WSN-MIB.txt` makes it possible to translate numeric OIDs to more human readable textual object identifiers. It gives meaning to the string of numbers used in the OID and also gives an easier way of remembering objects. If the MIB description file is placed in the appropriate location, it is used by the Net-SNMP applications and either numerical OIDs or textual object descriptors can be used to perform SNMP queries. This is expanded on in the next chapter. Net-SNMP is a collection of applications that can be run in a terminal on a computer and is used to send SNMP requests to SNMP agents.

In the `WSN-MIB.txt` file the five children and mote(1) subtree of the implemented MIB are defined by:

```
mote OBJECT IDENTIFIER ::= { wsn 1 }

status OBJECT IDENTIFIER ::= { mote 1 }
sensors OBJECT IDENTIFIER ::= { mote 2 }
external OBJECT IDENTIFIER ::= { mote 3 }
route OBJECT IDENTIFIER ::= { mote 4 }
stats OBJECT IDENTIFIER ::= { mote 5 }
```

After this, each object is defined separately under its specific section.

```
version OBJECT-TYPE
    SYNTAX      DisplayString (SIZE (0..255))
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "Version number for easy software identification."
    ::= { status 2 }
```

If objects are removed from, or added to the MIB, then the MIB description file should be updated to reflect these changes. If multiple MIBs are created and used on different nodes then each MIB should have a corresponding description file.

4.3.2 Statistics

In the `snmpMibP` component there are various objects that contain information and statistics about the node and these statistics can be accessed using the SNMP. Statistics such as the number of sent and received messages are kept and made available by the SNMP agent. Most of the statistics are provided by the `IPDispatchC` configuration file in the *blip* stack, these include information on routing, IP and ICMP messages. Statistics on the SNMP messages are kept by the `snmpMibP` component itself. The `snmpMibP` component provides a `Set` interface through which statistics can be kept on the messages generated by the main application code running on the node. The needed information is gathered by the handler function created for each object when a request for the information is received.

The `wsn.mote.route` section of the MIB provides access to routing information on the node. The statistics are provided by the *blip* stack and are wired through the `IPDispatchC` configurations file to `IPRoutingP`, which provides the `Statistics` interface for retrieving the information. The values available are `hop_limit`, `parent`, `parent_metric` and `parent_etx`. The exact significance of these values had to be deduced from the source code of the stack as it is not clearly documented anywhere. `hop_limit` points to the number of hops to the root node through the neighbour currently used for routing. `parent` gives the TOS_NODE_ID of the neighbour being routed through to the root node. TOS_NODE_ID is a global constant in TinyOS that is used to identify a node, it can also be seen as the node's address. The `parent_metric` field shows the additive path cost of the top entry in the routing table and `parent_etx` the additive path cost of the entry currently being used as the primary default route. These routing statistics were included to show that the SNMP agent can provide access to information kept in the stack itself and give some indication of what is happening with the routing. It is worth noting that it might be possible to modify the node's routing behaviour by using the SNMP and making the needed changes to *blip*.

The definitions of the structures containing the statistics provided by the *blip* stack can be found in the `Statistics.h` header file. These include `ip_statistics_t`, `route_-statistics_t`, `icmp_statistics_t` and `udp_statistics_t`.

In the MIB the `wsn.mote.stats` section provides information on the number of messages sent and received by the different protocols used by the node. Statistics on the number of received and transmitted messages are available for UDP, SNMP, IP, ICMP and also for the messages generated by the main application running on the node. There is a counter for the number of IP messages that the node forwards for other nodes. The number of UDP datagrams dropped due to checksum errors is also kept in the MIB. All the statistics for messages sent and received, except for the SNMP, are gathered from `blip`. Not all the statistics kept by the *blip* stack were included in the test MIB.

4.4 Applications

To demonstrate the functionality of the SNMPv1 agent implementation a few applications were created. The applications show how the external variables of the `snmpMibP` can be used in the main application on the node to control the LEDs and also timers running in the application. The applications also show that it is possible to implement an application on the WSN node alongside the SNMP agent. The applications can be set to use the low power mode provided by the *blip* stack and TinyOS.

4.4.1 The hopNode application

The `hopNode` application is a minimal application that demonstrates the SNMP agent. The main application code contains the minimum code needed to run the SNMP agent.

It also shows the use of the external variables in the MIB to control the LEDs, enabling or disabling them and changing the frequency at which they toggle through the `external.reportPeriod` objects. The `hopNode` application can generally be used on range extending nodes where the node is used to route messages for other nodes. The main application code does not generate any messages (unlike the `windStation` application discussed next), but it is possible to retrieve sensor readings using SNMP GetRequests or the SNMP Trap message set up for voltage readings. Figure 4.8 shows a wiring diagram of the hopNode application.

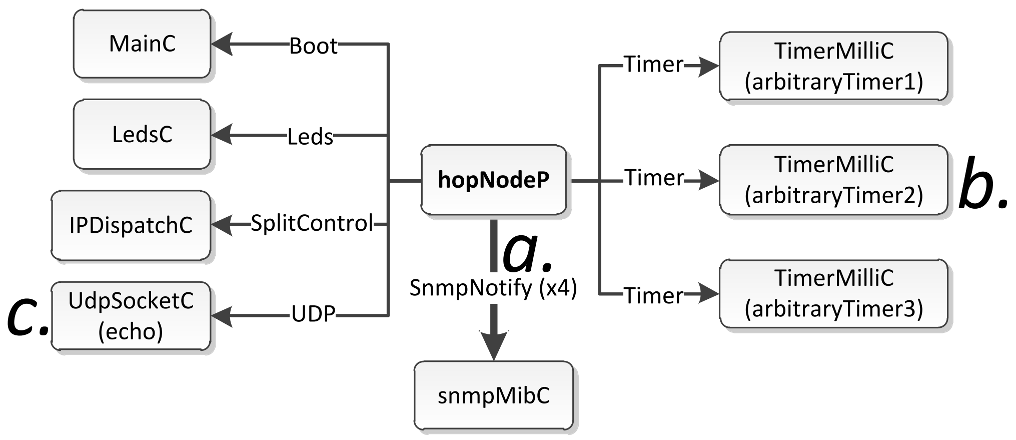


Figure 4.8: Wiring diagram of the hopNode application

The application uses the `SnmpNotify` interfaces (Fig. 4.8(*a.*)) provided by `snmpMibP` to control the LEDs on the node. There are three timers (Fig. 4.8(*b.*)) in the application that are started at boot time called *abritaryTimer1* to *abritaryTimer3*. When one of the `external.reportPeriod` objects is set to a new value through an SNMP SetRequest, an `snmpNotify` event (from the `SnmpNotify` interface) is signalled from the `snmpMibP` component and the corresponding *abritaryTimer* in the `hopNode` application is restarted with the new period value. The rate at which the LEDs on the node blink can therefore be changed using the SNMP. This is done by controlling variables in the main application code using the SNMP. An example of a handler function for an `SnmpNotify` event can be seen below, with the new period for the timer in the *val* variable.

```
event void snmpReportPeriod1.snmpNotify(uint32_t val)
{
  globalReportPeriodLocal1 = val;
  call arbitraryTimer1.startPeriodic(globalReportPeriodLocal1);
}
```

When one of the timers fire, a check is done to see if the LEDs are enabled. If the

LEDs are enabled, then the LED corresponding to the timer is toggled, and if the LEDs are disabled, then the function to turn off the specific LED is called.

If the `external.ledsEnable` object is changed in the MIB (implemented in the `snmpMib` block), then an `snmpNotify` event is generated and the LEDs are turned on or off according to the new value set.

The `hopNode` application also implements an echo service similar to that seen in the *UDPEcho* application in *blip* that echoes any message sent to the node back to the sender. This service uses the `UDP` interface (Fig. 4.8(*c.*)) provided by the `UdpSocketC` component as `echo`.

4.4.2 The windStation application

The `windStation` application contains more functionality than the `hopNode` application. This application generates messages containing sensor readings that are sent to the network management station as UDP packets, in addition to the ability to respond to SNMP messages. It also has the ability to control the period at which the timers in the application fire and control of the LEDs as shown in the previous application.

The application uses the `Set` interface (Fig. 4.9(*a.*)) provided by the `snmpMibC` component to update the statistics kept in the MIB on the number of messages sent and received by the application code. Figure 4.9 shows a wiring diagram of the windStation application.

A timer is used to trigger the gathering of the sensor readings (Fig. 4.9(*b.*)). Each time the timer fires a different reading is gathered until all sensor readings are gathered. The message with the sensor readings is then sent using the `UDP` interface provided through the `UdpSocketC` configuration (Fig. 4.9(*c.*)). The `windStation` application uses the `Read` interface to gather sensor readings from the TelosB onboard sensors and also from a connected anemometer through the use of the `windMeterC` component (Fig. 4.9(*d.*)). The `windStation` application also has the echo service implemented in the `hopNode` application (Fig. 4.9(*e.*))

The `windMeterP` component was created for an outdoor test deployment. The component deals with gathering sensor readings from a anemometer manufactured by Davis Instruments, which includes wind speed and direction.

The onboard sensors supply readings on supply voltage, temperature, humidity and light. Readings from the anemometer include wind direction and various speed read-

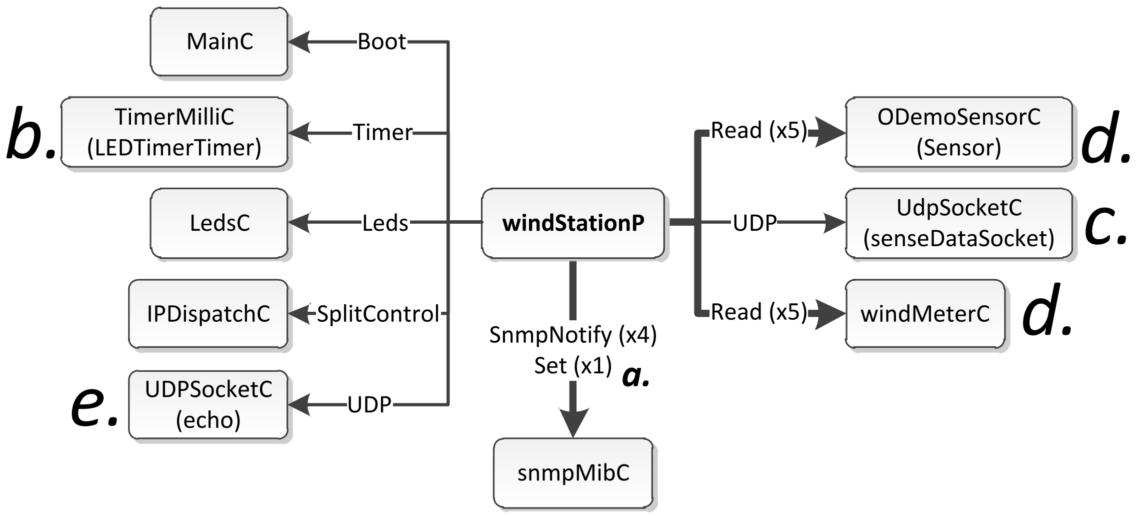


Figure 4.9: Wiring diagram of the windStation application

ings - minimum, maximum values over the time period measured and instantaneous and averaged speed values. The sensor readings are sent via UDP packet along with the TOS_NODE_ID of the node, software version number, period of the timer and an incrementing counter value used for packet identification.

The `windStation` application shows that it is possible to implement a relatively capable WSN application along with the SNMP agent on a TelosB WSN node. The application can also be configured to use the low power communication provided by the *blip* stack and TinyOS through the application `Makefile`, reducing the power consumed by the node.

4.4.3 Memory usage

This section characterises the created applications in terms of memory usage, both ROM (Flash) and RAM. The size of the applications largely depends on the function of the application and also on the MIB used. It is possible to implement a lightweight MIB to reduce the memory usage of an application, or implement a MIB that only contains the needed objects.

To get an overall impression of the size of the compiled code, Table 4.2 was constructed. This table serves as a general indicator of the sizes of compiled applications using the SNMPv1 agent. The TelosB platform contains a MSP430F1611 microcontroller from Texas Instruments with 48k bytes flash and 10k bytes RAM. The two created applications `hopNode` and `windStation` along with a demonstration application called `UDPEcho`, included with *blip*, are shown. The `Blink` application is included in TinyOS and only flashes the node's LEDs. `Blink` does not use *blip* and is arguably one of the simplest TinyOS applications. It therefore gives an indication of the memory usage of some of the smallest TinyOS applications, although not very functional. The table gives an indication of the size of applications using the SNMP software agent relative to other applications.

Table 4.2: RAM and ROM usage, in bytes, of applications with and without Low Power Listen (LPL)

Application	**ROM**	**RAM**
Blink	2648	54
UDPEcho	29486	5062
UDPEcho (LPL)	31150	5124
hopNode	39712	5960
hopNode(LPL)	41396	6022
windStation	43300	6030
windStation(LPL)	44970	6090

4.5 Summary

This chapter covered the implementation of the SNMPv1 software agent and also the test applications created. The organisation of the SNMP agent was shown along with relevant code snippets to aid explanation. The created applications, their use and purpose were discussed and lastly some characteristics of the applications in the form of memory usage were also shown. In the following chapter the operation of the SNMP agent is shown and validated. Results obtained with the packages Cacti and Nagios will also be shown.

Chapter 5

Testing and evaluation

The SNMP software agent was tested at two levels. The first being general testing from a workstation, which also acts as the edge router. The workstation has two WSN nodes attached, one node serves as the IEEE 802.15.4 interface (programmed with a radio-serial bridge) of the edge router and a second node running the code under test. This testing was generally done from the terminal using the commands provided by Net-SNMP. This was the type of testing that could be done quickly and continuously during development to test the general functioning of the software agent.

The second level of testing was done using an indoor testbed and multiple nodes, this facilitated the use of the network monitoring software. This testing shows the functioning of the software agent over a longer time period. It also shows the functioning of the software agent with the selected network monitoring packages.

5.1 Workstation

Workstation testing is testing that could be done quickly while busy with the development of the SNMP agent. In general this consisted of two nodes connected to a PC with one node acting as the IEEE 802.15.4 interface and the second as the node running the software under test. The PC was used for testing/debugging and acted as the edge router for the *blip* network, seen in Figure 5.1. The *blip* routing driver uses the IEEE 802.15.4 interface node to communicate with the node used for testing. The USB connection between the nodes and the PC is used to power the nodes and for programming one of the nodes with the software to be tested. This section will mainly consist of tests done using applications from the Net-SNMP package.

A *blip* 6LoWPAN network consists of a few different elements. The first is the edge router, which typically consists of a Linux device (the PC in Figure 5.1) with a WSN node connected to it that acts as its IEEE 802.15.4 interface device. This node is programmed with a radio-serial bridge application. The edge router runs the *blip* routing driver program. This edge router enables the user to communicate with the *blip* nodes in the WSN or, for this test, the one node running the application code under test.

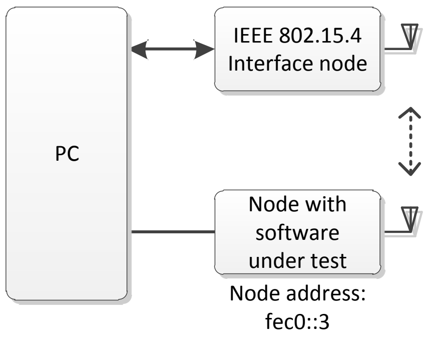


Figure 5.1: Workstation testing setup

Initially the Low Power Listen (LPL) functionality of TinyOS was not used, but later versions of the agent were tested with the LPL function enabled. Very early on in testing, the TinyOS *printf* library was also used to debug code. After initial testing, Wireshark was used and set up to monitor packets passing through the `tun0` interface created by the routing driver. Test messages were sent using `snmpget` and `snmpset` with the responses being evaluated using Wireshark and through the output of the Net-SNMP commands.

5.1.1 Preliminary blip tests

The first test that can be done is to send an ICMP (Internet Control Message Protocol) echo request (ping) to the node. This is also a good way to determine if a node is up or available during normal operation. The time it takes for a request to travel to the node and back is affected by a number of things, two of which are the path to the node (number of hops) and whether LPL is used. Below is an echo request sent to a network that does not use LPL.

```
$ ping6 -c 4 -i 2 fec0::3
PING fec0::3(fec0::3) 56 data bytes
64 bytes from fec0::3: icmp_seq=1 ttl=65 time=74.9 ms
64 bytes from fec0::3: icmp_seq=2 ttl=65 time=81.6 ms
64 bytes from fec0::3: icmp_seq=3 ttl=65 time=79.9 ms
64 bytes from fec0::3: icmp_seq=4 ttl=65 time=78.9 ms
```

```
--- fec0::3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 6004ms
rtt min/avg/max/mdev = 74.906/78.882/81.680/2.499 ms
```

Of the programs and commands used, the Linux `ping6` command for sending ICMP ECHO_ REQUESTs was arguably used the most. In its simplest form the command only takes as an argument the IPv6 node address of the node to be queried. There are options that can be used to make the command better behaved in networks with longer round trip times (RTT) by not sending requests quicker than a node can respond. The round trip times for messages in WSNs are generally higher than on conventional IP networks, with even longer round trip times if LPL is used. The example below shows the use of the `-c` and `-i` options to control the number of ECHO_REQUEST packets sent (`-c`) and the interval between packets (`-i`), four packets are sent at an interval of two seconds to the address `fec0::3`.

```
$ ping6 -c 4 -i 2 fec0::3
```

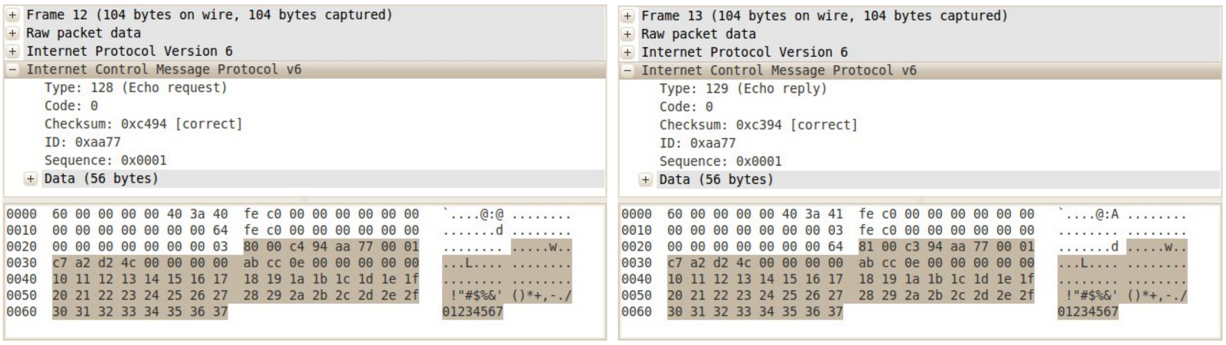
A network that uses LPL has considerably higher and more variable round trip times compared to the non-LPL network, as can be seen from the output below. This is due to how the LPL MAC layer operates and duty cycles the radio on the node. It is worth noting the higher times when setting up notifications in software like Nagios, whose defaults are set according to conventional networks. The increase in round trip time also affects SNMP messages and should be kept in mind when sending messages.

```
$ ping6 -c 4 -i 2 fec0::3
PING fec0::3(fec0::3) 56 data bytes
64 bytes from fec0::3: icmp_seq=1 ttl=65 time=251 ms
64 bytes from fec0::3: icmp_seq=2 ttl=65 time=410 ms
64 bytes from fec0::3: icmp_seq=3 ttl=65 time=581 ms
64 bytes from fec0::3: icmp_seq=4 ttl=65 time=236 ms
--- fec0::3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 6004ms
rtt min/avg/max/mdev = 236.710/369.947/581.579/139.893 ms
```

Figure 5.2 shows an ICMP echo request and the response from the WSN node as viewed from and decoded by Wireshark.

Wireshark is a multi-platform, open source (under the GNU General Public License) packet analyser that can be used for network analysis and troubleshooting and uses the *pcap* library to capture network packets. Wireshark was formerly known as Ethereal.

Wireshark, as seen in Figure 5.3, was extensively used to debug the SNMPv1 agent during development. This could be done with no modifications or additions to Wireshark

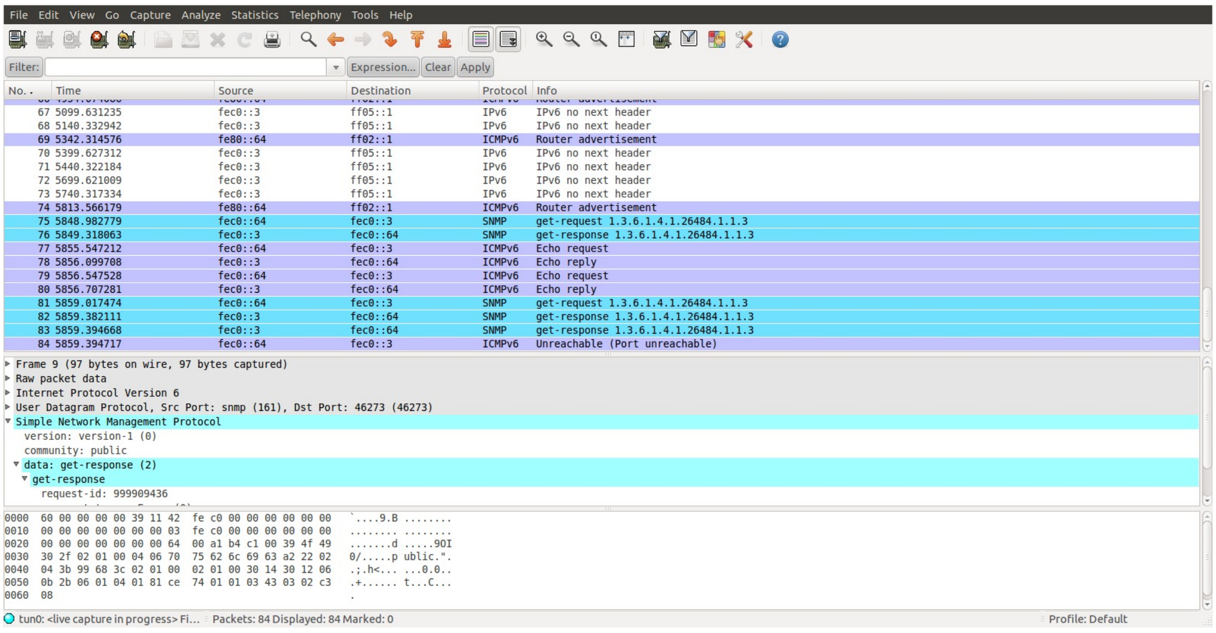


```
+ Frame 12 (104 bytes on wire, 104 bytes captured)
+ Raw packet data
+ Internet Protocol Version 6
- Internet Control Message Protocol v6
    Type: 128 (Echo request)
    Code: 0
    Checksum: 0xc494 [correct]
    ID: 0xaa77
    Sequence: 0x0001
  + Data (56 bytes)

0000  60 00 00 00 00 40 3a 40  fe c0 00 00 00 00 00 00   `....@:@ ........
0010  00 00 00 00 00 00 00 64  fe c0 00 00 00 00 00 00   .......d ........
0020  00 00 00 00 00 00 00 03  80 00 c4 94 aa 77 00 01   ........ .....w..
0030  c7 a2 d2 4c 00 00 00 00  ab cc 0e 00 00 00 00 00   ...L.... ........
0040  10 11 12 13 14 15 16 17  18 19 1a 1b 1c 1d 1e 1f   ........ ........
0050  20 21 22 23 24 25 26 27  28 29 2a 2b 2c 2d 2e 2f    !"#$%&' ()*+,-./
0060  30 31 32 33 34 35 36 37                            01234567
```

```
+ Frame 13 (104 bytes on wire, 104 bytes captured)
+ Raw packet data
+ Internet Protocol Version 6
- Internet Control Message Protocol v6
    Type: 129 (Echo reply)
    Code: 0
    Checksum: 0xc394 [correct]
    ID: 0xaa77
    Sequence: 0x0001
  + Data (56 bytes)

0000  60 00 00 00 00 40 3a 41  fe c0 00 00 00 00 00 00   `....@:A ........
0010  00 00 00 00 00 00 00 03  fe c0 00 00 00 00 00 00   ........ ........
0020  00 00 00 00 00 00 00 64  81 00 c3 94 aa 77 00 01   .......d .....w..
0030  c7 a2 d2 4c 00 00 00 00  ab cc 0e 00 00 00 00 00   ...L.... ........
0040  10 11 12 13 14 15 16 17  18 19 1a 1b 1c 1d 1e 1f   ........ ........
0050  20 21 22 23 24 25 26 27  28 29 2a 2b 2c 2d 2e 2f    !"#$%&' ()*+,-./
0060  30 31 32 33 34 35 36 37                            01234567
```

Figure 5.2: ICMP echo request and response messages in Wireshark

because the packets passing through the `tun0` interface created by the *blip* routing driver are IPv6 packets. `tcpdump` was used to monitor traffic from the WSN remotely from a terminal. Similar to Wireshark, `tcpdump` is a command line packet analyser released under the BSD license. The contents of received network packets are printed to the command line using *pcap*. The interface that is listened to, in this case `tun0`, is specified by the `-i` option.



```
No. .  Time            Source      Destination  Protocol  Info
    67 5099.631235     fec0::3     ff05::1      IPv6      IPv6 no next header
    68 5140.332942     fec0::3     ff05::1      IPv6      IPv6 no next header
    69 5342.314576     fe80::64    ff02::1      ICMPv6    Router advertisement
    70 5399.627312     fec0::3     ff05::1      IPv6      IPv6 no next header
    71 5440.322184     fec0::3     ff05::1      IPv6      IPv6 no next header
    72 5699.621009     fec0::3     ff05::1      IPv6      IPv6 no next header
    73 5740.317334     fec0::3     ff05::1      IPv6      IPv6 no next header
    74 5813.566179     fe80::64    ff02::1      ICMPv6    Router advertisement
    75 5848.982779     fec0::64    fec0::3      SNMP      get-request 1.3.6.1.4.1.26484.1.1.3
    76 5849.318063     fec0::3     fec0::64     SNMP      get-response 1.3.6.1.4.1.26484.1.1.3
    77 5855.547212     fec0::64    fec0::3      ICMPv6    Echo request
    78 5856.099708     fec0::3     fec0::64     ICMPv6    Echo reply
    79 5856.547528     fec0::64    fec0::3      ICMPv6    Echo request
    80 5856.707281     fec0::3     fec0::64     ICMPv6    Echo reply
    81 5859.017474     fec0::64    fec0::3      SNMP      get-request 1.3.6.1.4.1.26484.1.1.3
    82 5859.382111     fec0::3     fec0::64     SNMP      get-response 1.3.6.1.4.1.26484.1.1.3
    83 5859.394668     fec0::3     fec0::64     SNMP      get-response 1.3.6.1.4.1.26484.1.1.3
    84 5859.394717     fec0::64    fec0::3      ICMPv6    Unreachable (Port unreachable)

▸ Frame 9 (97 bytes on wire, 97 bytes captured)
▸ Raw packet data
▸ Internet Protocol Version 6
▸ User Datagram Protocol, Src Port: snmp (161), Dst Port: 46273 (46273)
▾ Simple Network Management Protocol
    version: version-1 (0)
    community: public
  ▾ data: get-response (2)
    ▾ get-response
        request-id: 999909436

0000  60 00 00 00 00 39 11 42  fe c0 00 00 00 00 00 00   `....9.B ........
0010  00 00 00 00 00 00 00 03  fe c0 00 00 00 00 00 00   ........ ........
0020  00 00 00 00 00 00 00 64  00 a1 b4 c1 00 39 4f 49   .......d .....9OI
0030  30 2f 02 01 00 04 06 70  75 62 6c 69 63 a2 22 02   0/.....p ublic.".
0040  04 3b 99 68 3c 02 01 00  02 01 00 30 14 30 12 06   .;.h<... ...0.0..
0050  0b 2b 06 01 04 01 81 ce  74 01 01 03 43 03 02 c3   .+...... t...C...
0060  08                                                 .

tun0: <live capture in progress> Fi...   Packets: 84 Displayed: 84 Marked: 0        Profile: Default
```

Figure 5.3: Wireshark packet analyser in operation

```
$ sudo tcpdump -i tun0
tcpdump: WARNING: tun0: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on tun0, link-type RAW (Raw IP), capture size 65535 bytes
19:33:40.759057 IP6 fec0::64 > fec0::3: ICMP6, echo request, seq 1, length 64
19:33:40.918137 IP6 fec0::3 > fec0::64: ICMP6, echo reply, seq 1, length 64
19:33:42.761458 IP6 fec0::64 > fec0::3: ICMP6, echo request, seq 2, length 64
19:33:43.088352 IP6 fec0::3 > fec0::64: ICMP6, echo reply, seq 2, length 64
19:33:49.428174 IP6 fe80::3 > ip6-allrouters: ICMP6, router solicitation, length 8
19:33:49.428860 IP6 fe80::64 > ip6-allnodes: ICMP6, router advertisement, length 56
```

To a lesser extent the `tracert6` command was also used. It gives an indication of what path a packet takes through the network to its destination. The `tracert6` program does this by sending ICMP ECHO_REQUEST packets while incrementing the hop limit of each packet until the destination is reached. The `-q` option can be used to change the number of probes sent to each hop, which is set to three by default.

```
$ tracert6 -q 2 fec0::d
```

The trip times between hops seen in `tracert6` will vary according to the network, and whether LPL is used, because `tracert6` makes use of ICMP echo requests to determine the route to a node. The functioning of the MAC layer on each node can also have an effect on RTT. This specific test was done on the testbed. Below it can be seen that for that specific period in time node `fec0::d` used `fec0::8` to communicate with the edge router `fec0::64`. The round trip times for each probe sent to every hop can be seen after the address in the output of the command. For example, the first probe to the first hop (`fec0::8`) has a RTT of 276.923 ms.

```
$ tracert6 fec0::d
traceroute to fec0::d (fec0::d) from fec0::64, 30 hops max, 60 byte packets
 1  fec0::8 (fec0::8)  276.923 ms  78.991 ms  182.260 ms
 2  fec0::d (fec0::d)  625.612 ms  709.606 ms  184.124 ms
```

Information on the path a packet takes over multiple hops to its destination can also be retrieved by using the `routes` command from the console provided by the *blip* `ip-driver` program (the routing driver). The `ip-driver` program is part of *blip* and is the link between the WSN and conventional LAN (Local Area Network). It is also responsible for creating the tunnel network interface, routing packets in the WSN and between the WSN and the LAN. The `ip-driver` routing driver provides a telnet console server running on port 6106 through which the driver can be queried or configured. The `help` command lists the available commands for use in the console.

A tool that was found to be useful was GNU Screen through Byobu. It is particularly useful when running the *blip* routing driver. A number of terminal windows can be set up in Byobu and the session then detached while the driver is left running. The session can then be continued remotely to check on the running software and network. Windows for the driver, programming of nodes, SNMP requests and pinging of nodes were created. These windows and the command history for each window could quickly be reused to perform tasks.

GNU Screen can be thought of as a window manager for the terminal that enables multiple separate terminal sessions from a single terminal window. Each session has a scrollback history buffer and the user is able to copy text from one session to another.

A terminal session can be detached from the current terminal window and then later attached to another terminal window without interrupting the application running in that terminal session. Byobu is an enhancement script that launches GNU Screen in the Byobu configuration, which shows system information and notifications not present in GNU Screen by default. Byobu, in essence, makes GNU Screen look more user friendly with more information visible like named tabs.

5.1.2 SNMP tests

The `snmpget` and `snmpset` Net-SNMP applications from the *snmp* package in Ubuntu can be used to manually interrogate SNMP software agents. The use of these applications is relatively straightforward, although there are some options that can be used that are better suited to use over a WSN.

The `snmpget` application generates an SNMP *GetRequest* message for the wanted object at the address specified when the application is run. The software agent on the network node processes the message and then responds with an appropriate SNMP *GetResponse* message. The response is then displayed by the application.

```
$ snmpget -v 1 -c public UDP6:[fec0::2] 1.3.6.1.4.1.26484.1.1.1
```

The SNMP version number is specified using the `-v 1` option, while `-c public` specifies the community string being used. The location of the MIB descriptor file can be indicated using the `-m` option as shown:

```
-m <PATH TO>/components/snmpMib/WSN-MIB.txt
```

This enables the translation of the OIDs from numeric form to textual object identifiers using the MIB description file specified. This can however be cumbersome, another option is to create an `snmp.conf` file. A `.snmp` directory is created in the user's home directory and the `snmp.conf` and MIB files placed in the directory. The `snmp.conf` file contains the following line:

```
mibs +/home/<user>/.snmp/WSN-MIB.txt
```

It should be noted that the destination of the request is specified as `UDP6:[fec0::2]` with the last argument to the `snmpget` application being the OID `1.3.6.1.4.1.26484.1.1.1`. The `UDP6:[]` specifies the transport protocol to be used, followed by the `fec0::2` IPv6 address, where the final `2` from the address relates to the TOS_NODE_ID given to the node when programmed. The example shows the arguments used to gather the *sysDescr* object from a node with the TOS_NODE_ID of 2. TOS_NODE_ID is a global constant in

TinyOS that is used to identify a node. The assigned IPv6 address is based on the node's TOS_NODE_ID.

Using the above `snmpget` application, information can be retrieved from a network node. To change information on a node the `snmpset` application is used. The `snmpset` application generates an SNMP *SetRequest* message that is sent to the network node, the node processes the message and responds with an SNMP *GetResponse* message.

The options used are similar to that of the `snmpget` application with the `-v`, `-c` and `-m` options present. The address and OID arguments are also similar. The value to which the variable, indicated by the OID, should be set and its *type* is specified as the last two arguments to the application.

```
$ snmpset -v 1 -c public UDP6:[fec0::2] 1.3.6.1.4.1.26484.1.3.1 u "0"
```

In the above example the type *u* indicates an unsigned integer with the value being set to zero. In this case the variable `ledsEnable` is set to zero, disabling any LEDs on the node.

For both the applications it is possible to set the number of retries (`-r`) to be used in a request and also the timeout (`-t`) between retries. This is useful when low power strategies, like TinyOS's low power listening feature is used, which can cause the message latency to become considerably higher than what is normally expected on a conventional network. Setting a higher timeout between requests gives the network enough time to propagate the message and the node enough time to respond, without flooding the network with multiple unnecessary messages.

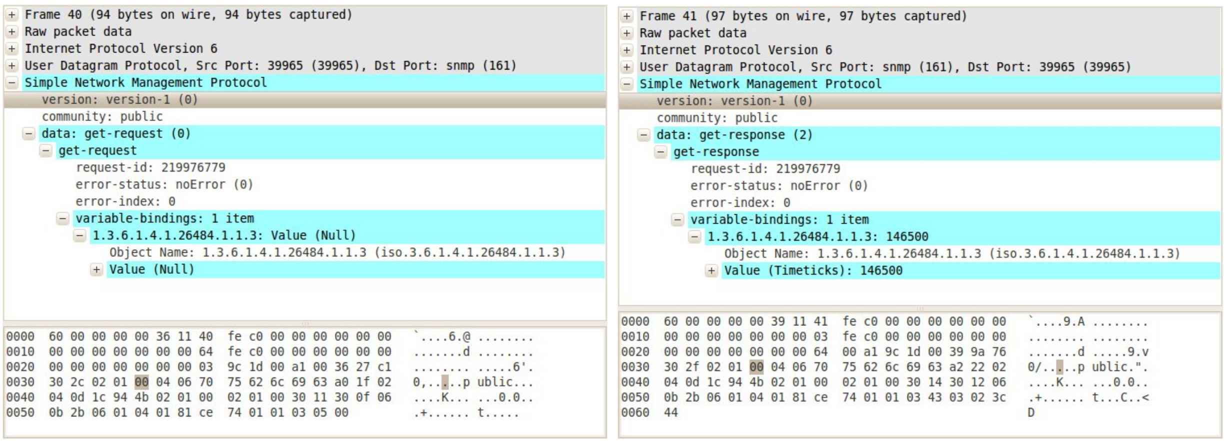
It is possible to specify the required object in a number of different ways when using `snmpget` and `snmpset`. As shown above, the numerical OID can be used to specify the particular object, but it is also possible to use the textual object descriptor of the object or combinations of both numerical and textual. Using the following as an argument to `snmpget` or `snmpset` will yield the same results:

```
1.3.6.1.4.1.26484.1.1.3
1.3.6.1.4.1.wsn.mote.status.sysUpTime
iso.org.dod.internet.private.enterprises.wsn.mote.status.sysUpTime
WSN-MIB::mote.status.sysUpTime
WSN-MIB::mote.status.3
wsn.mote.status.sysUpTime
wsn.mote.status.3
```

An example of requesting the node's uptime using `snmpget` is shown below. If the OID is correct, the node supports the MIB and no errors occurred, then the command exits with the retrieved value. As output the raw timeticks value is given, 146500, and the interpreted value in the form hours:minutes:seconds.fractions of a second. The node's uptime was verified by monitoring uptime returned by SNMP against an external clock.

```
$ snmpget -v 1 -c public UDP6:[fec0::3] mote.status.sysUpTime
WSN-MIB::sysUpTime = Timeticks: (146500) 0:24:25.00
```

Figure 5.4 shows the SNMP GetRequest generated by the `snmpget` command and the reply from the WSN node as seen in Wireshark.



```
+ Frame 40 (94 bytes on wire, 94 bytes captured)
+ Raw packet data
+ Internet Protocol Version 6
+ User Datagram Protocol, Src Port: 39965 (39965), Dst Port: snmp (161)
- Simple Network Management Protocol
    version: version-1 (0)
    community: public
  - data: get-request (0)
    - get-request
        request-id: 219976779
        error-status: noError (0)
        error-index: 0
      - variable-bindings: 1 item
        - 1.3.6.1.4.1.26484.1.1.3: Value (Null)
            Object Name: 1.3.6.1.4.1.26484.1.1.3 (iso.3.6.1.4.1.26484.1.1.3)
          + Value (Null)

0000  60 00 00 00 00 36 11 40  fe c0 00 00 00 00 00 00   `....6.@ ........
0010  00 00 00 00 00 00 00 64  fe c0 00 00 00 00 00 00   .......d ........
0020  00 00 00 00 00 00 00 03  9c 1d 00 a1 00 36 27 c1   ........ .....6'.
0030  30 2c 02 01 00 04 06 70  75 62 6c 69 63 a0 1f 02   0,.....p ublic...
0040  04 0d 1c 94 4b 02 01 00  02 01 00 30 11 30 0f 06   ....K... ...0.0..
0050  0b 2b 06 01 04 01 81 ce  74 01 01 03 05 00         .+...... t.....
```

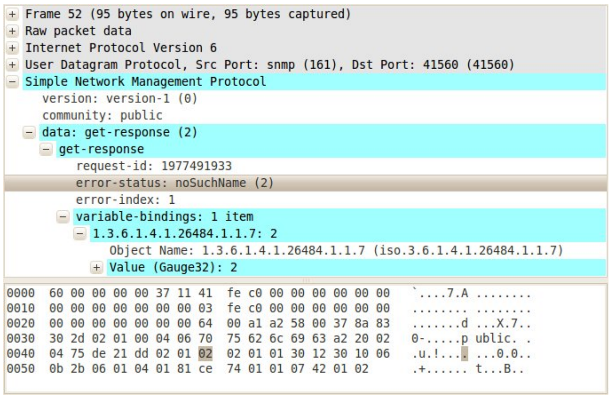
```
+ Frame 41 (97 bytes on wire, 97 bytes captured)
+ Raw packet data
+ Internet Protocol Version 6
+ User Datagram Protocol, Src Port: snmp (161), Dst Port: 39965 (39965)
- Simple Network Management Protocol
    version: version-1 (0)
    community: public
  - data: get-response (2)
    - get-response
        request-id: 219976779
        error-status: noError (0)
        error-index: 0
      - variable-bindings: 1 item
        - 1.3.6.1.4.1.26484.1.1.3: 146500
            Object Name: 1.3.6.1.4.1.26484.1.1.3 (iso.3.6.1.4.1.26484.1.1.3)
          + Value (Timeticks): 146500

0000  60 00 00 00 00 39 11 41  fe c0 00 00 00 00 00 00   `....9.A ........
0010  00 00 00 00 00 00 00 03  fe c0 00 00 00 00 00 00   ........ ........
0020  00 00 00 00 00 00 00 64  00 a1 9c 1d 00 39 9a 76   .......d .....9.v
0030  30 2f 02 01 00 04 06 70  75 62 6c 69 63 a2 22 02   0/.....p ublic.".
0040  04 0d 1c 94 4b 02 01 00  02 01 00 30 14 30 12 06   ....K... ...0.0..
0050  0b 2b 06 01 04 01 81 ce  74 01 01 03 43 03 02 3c   .+...... t...C..<
0060  44                                                 D
```

Figure 5.4: SNMP GetRequest and GetResponse in Wireshark

If the node is queried with an OID that it does not have in its MIB, then it replies with an error message. This can be seen below when the node is queried with an OID that does not exist in the `status` section of the MIB. The GetRequest that is sent to the node is similar to the GetRequest seen above with the exception of the OID. The output from the `snmpget` command states that an error has occurred and also the reason for the error.

```
$ snmpget -v 1 -c public UDP6:[fec0::3] mote.status.7
Error in packet
Reason: (noSuchName) There is no such variable name in this MIB.
Failed object: WSN-MIB::status.7
```

Figure 5.5 shows the GetResponse message containing the error message as it was sent from the WSN node and captured in Wireshark. The error-status field is set to 0x02 to indicate noSuchName and the error-index set to 0x01 to indicate that it is the first item in the VarBindList that created the error.



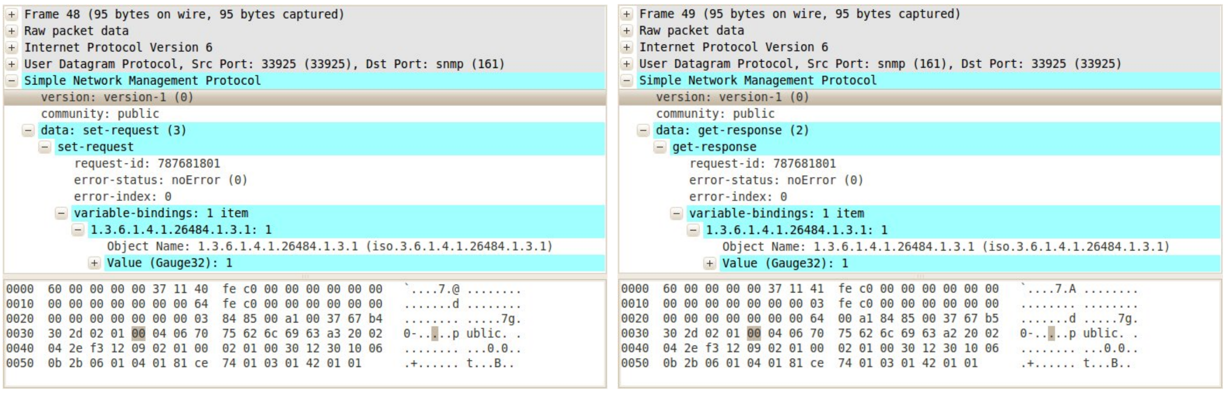
```
+ Frame 52 (95 bytes on wire, 95 bytes captured)
+ Raw packet data
+ Internet Protocol Version 6
+ User Datagram Protocol, Src Port: snmp (161), Dst Port: 41560 (41560)
- Simple Network Management Protocol
    version: version-1 (0)
    community: public
  - data: get-response (2)
    - get-response
        request-id: 1977491933
        error-status: noSuchName (2)
        error-index: 1
      - variable-bindings: 1 item
        - 1.3.6.1.4.1.26484.1.1.7: 2
            Object Name: 1.3.6.1.4.1.26484.1.1.7 (iso.3.6.1.4.1.26484.1.1.7)
          + Value (Gauge32): 2

0000  60 00 00 00 00 37 11 41  fe c0 00 00 00 00 00 00   `....7.A ........
0010  00 00 00 00 00 00 00 03  fe c0 00 00 00 00 00 00   ........ ........
0020  00 00 00 00 00 00 00 64  00 a1 a2 58 00 37 8a 83   .......d ...X.7..
0030  30 2d 02 01 00 04 06 70  75 62 6c 69 63 a2 20 02   0-.....p ublic. .
0040  04 75 de 21 dd 02 01 02  02 01 01 30 12 30 10 06   .u.!.... ...0.0..
0050  0b 2b 06 01 04 01 81 ce  74 01 01 07 42 01 02      .+...... t...B..
```

Figure 5.5: SNMP GetResponse with error message

Some variables in the MIB on the node can be changed by using the `snmpset` command. The node responds with an SNMP GetResponse message. The structure this message is identical to the previously shown messages and contains the value that the variable was set to, as can be seen below.

```
$ snmpset -v 1 -c public UDP6:[fec0::3] mote.external.ledsEnable u "1"
WSN-MIB::ledsEnable = Gauge32: 1
```

The contents, as viewed in Wireshark, of the SetRequest and the GetResponse from the WSN node can be seen in Figure 5.6. In this example the LEDs on the node can be turned on by writing the value one to the `ledsEnable` variable in the MIB.



```
+ Frame 48 (95 bytes on wire, 95 bytes captured)
+ Raw packet data
+ Internet Protocol Version 6
+ User Datagram Protocol, Src Port: 33925 (33925), Dst Port: snmp (161)
- Simple Network Management Protocol
    version: version-1 (0)
    community: public
  - data: set-request (3)
    - set-request
        request-id: 787681801
        error-status: noError (0)
        error-index: 0
      - variable-bindings: 1 item
        - 1.3.6.1.4.1.26484.1.3.1: 1
            Object Name: 1.3.6.1.4.1.26484.1.3.1 (iso.3.6.1.4.1.26484.1.3.1)
          + Value (Gauge32): 1

0000  60 00 00 00 00 37 11 40  fe c0 00 00 00 00 00 00   `....7.@ ........
0010  00 00 00 00 00 00 00 64  fe c0 00 00 00 00 00 00   .......d ........
0020  00 00 00 00 00 00 00 03  84 85 00 a1 00 37 67 b4   ........ .....7g.
0030  30 2d 02 01 00 04 06 70  75 62 6c 69 63 a3 20 02   0-.....p ublic. .
0040  04 2e f3 12 09 02 01 00  02 01 00 30 12 30 10 06   ........ ...0.0..
0050  0b 2b 06 01 04 01 81 ce  74 01 03 01 42 01 01      .+...... t...B..
```

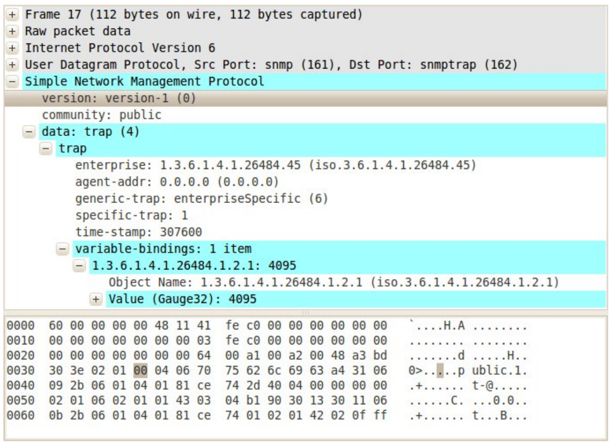
```
+ Frame 49 (95 bytes on wire, 95 bytes captured)
+ Raw packet data
+ Internet Protocol Version 6
+ User Datagram Protocol, Src Port: snmp (161), Dst Port: 33925 (33925)
- Simple Network Management Protocol
    version: version-1 (0)
    community: public
  - data: get-response (2)
    - get-response
        request-id: 787681801
        error-status: noError (0)
        error-index: 0
      - variable-bindings: 1 item
        - 1.3.6.1.4.1.26484.1.3.1: 1
            Object Name: 1.3.6.1.4.1.26484.1.3.1 (iso.3.6.1.4.1.26484.1.3.1)
          + Value (Gauge32): 1

0000  60 00 00 00 00 37 11 41  fe c0 00 00 00 00 00 00   `....7.A ........
0010  00 00 00 00 00 00 00 03  fe c0 00 00 00 00 00 00   ........ ........
0020  00 00 00 00 00 00 00 64  00 a1 84 85 00 37 67 b5   .......d .....7g.
0030  30 2d 02 01 00 04 06 70  75 62 6c 69 63 a2 20 02   0-.....p ublic. .
0040  04 2e f3 12 09 02 01 00  02 01 00 30 12 30 10 06   ........ ...0.0..
0050  0b 2b 06 01 04 01 81 ce  74 01 03 01 42 01 01      .+...... t...B..
```

Figure 5.6: SNMP SetRequest and GetResponse in Wireshark

It is possible to configure the WSN node to send SNMP Trap messages. This is done with the `trapState` object in the MIB. The frequency at which these messages are sent is controlled by the `trapPeriod` object. Table 4.1 in the previous chapter shows the values to which the `trapState` object can be set. Below, the `trapState` object is set to two and the WSN node responds with an SNMP GetResponse message, as was seen earlier.

```
$ snmpset -v 1 -c public UDP6:[fec0::3] mote.status.trapState u "2"
WSN-MIB::trapState = Gauge32: 2
```

The WSN node then starts generating SNMP Trap messages at intervals that are determined by the `trapPeriod` object. An SNMP Trap message sent by the WSN node can be seen in Figure 5.7, as captured by Wireshark.



```
+ Frame 17 (112 bytes on wire, 112 bytes captured)
+ Raw packet data
+ Internet Protocol Version 6
+ User Datagram Protocol, Src Port: snmp (161), Dst Port: snmptrap (162)
- Simple Network Management Protocol
    version: version-1 (0)
    community: public
  - data: trap (4)
    - trap
        enterprise: 1.3.6.1.4.1.26484.45 (iso.3.6.1.4.1.26484.45)
        agent-addr: 0.0.0.0 (0.0.0.0)
        generic-trap: enterpriseSpecific (6)
        specific-trap: 1
        time-stamp: 307600
      - variable-bindings: 1 item
        - 1.3.6.1.4.1.26484.1.2.1: 4095
            Object Name: 1.3.6.1.4.1.26484.1.2.1 (iso.3.6.1.4.1.26484.1.2.1)
          + Value (Gauge32): 4095

0000  60 00 00 00 00 48 11 41  fe c0 00 00 00 00 00 00   `....H.A ........
0010  00 00 00 00 00 00 00 03  fe c0 00 00 00 00 00 00   ........ ........
0020  00 00 00 00 00 00 00 64  00 a1 00 a2 00 48 a3 bd   .......d .....H..
0030  30 3e 02 01 00 04 06 70  75 62 6c 69 63 a4 31 06   0>.....p ublic.1.
0040  09 2b 06 01 04 01 81 ce  74 2d 40 04 00 00 00 00   .+...... t-@.....
0050  02 01 06 02 01 01 43 03  04 b1 90 30 13 30 11 06   ......C. ...0.0..
0060  0b 2b 06 01 04 01 81 ce  74 01 02 01 42 02 0f ff   .+...... t...B...
```

Figure 5.7: SNMP Trap message

The `trapPeriod` object has a default value of 10 000 milliseconds (10 seconds). This can be verified from the output of `tcpdump`.

```
22:06:34.825263 IP6 fec0::3.snmp > fec0::64.snmp-trap:  Trap(49)  E:26484.45 0.0.0.0 enterpriseSpecific
          s=1 172800 E:26484.1.2.1=4095
22:06:44.593062 IP6 fec0::3.snmp > fec0::64.snmp-trap:  Trap(49)  E:26484.45 0.0.0.0 enterpriseSpecific
          s=1 173800 E:26484.1.2.1=4095
22:06:54.357963 IP6 fec0::3.snmp > fec0::64.snmp-trap:  Trap(49)  E:26484.45 0.0.0.0 enterpriseSpecific
          s=1 174800 E:26484.1.2.1=4095
```

The workstation testing verifies that the SNMPv1 software agent and the node it is running on behaves as expected. The agent was verified using the Net-SNMP package with the agent being able to interpret SNMP messages sent to it and replying with messages that could be translated by the Net-SNMP applications.

5.2 Testbed

After the SNMPv1 agent was validated and tested under the ‘workstation’ tests, the code was deployed on the constructed testbed for further testing. This type of testing was typically done over a longer period of time and the software Cacti and Nagios could be used. The data gathered using the software tools and the SNMPv1 agent also became more interesting once multiple nodes were involved.

The testbed that was constructed and used is described in more detail in Appendix A. It consists of a PC and thirteen Tmote Connects from Sentilla (previously Moteiv) each with at least one TelosB platform wireless sensor node. The Tmote Connect is a Linksys NSLU2 with custom firmware. The NSLU2 is a network attached storage device with ethernet and two USB ports. The PC is used to program the nodes, often remotely, through the Tmote Connects using `netbsl`. `netbsl` is a script that can be used to remotely program a wireless sensor node connected to a Tmote Connect using Netcat, a networking utility that is able to read and write data over network connections.

Initially, Harvard's Motelab software was to be used on the testbed, but the installation and setup of the software was problematic. The Motelab software was abandoned and custom scripts were created to program the testbed using `netbsl`.

A method to log messages to a MySQL database was created to log the UDP messages, containing physical sensor readings, generated by the `windStation` application. In order to visualise the logged data, graphing scripts were created using PHP.

The testbed hardware and PC were only used to program and supply power to the nodes. A second PC, in addition to the testbed PC, acted as the edge router with an IEEE 802.15.4 interface node and the *blip* routing driver. This PC was also home to the network monitoring software (NMS), MySQL database and PHP graphing scripts. Figure 5.8 shows the physical layout of the testbed with each node's TOS_NODE_ID shown and the edge router and interface node shown with the designator `IF`. Figure 5.9 shows a diagram of the logical layout of the testbed hardware. Figure 5.10 shows a diagram of the PC acting as the edge router and also used for testing. This setup keeps the testbed hardware separate from the 6LoWPAN edge router. The programming of the nodes is covered in detail in Appendix A.

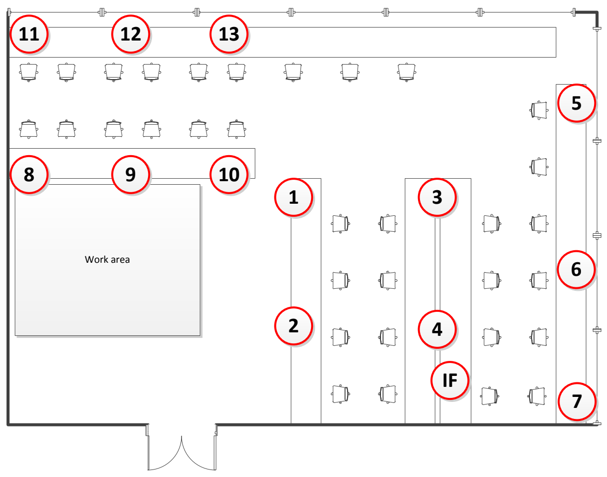


Figure 5.8: Floor plan of testbed and node location

Not all the nodes in the testbed were used. Node 13 was selected to demonstrate the PHP graphing and Cacti. This node had the IPv6 address `fec0::d` and was referred

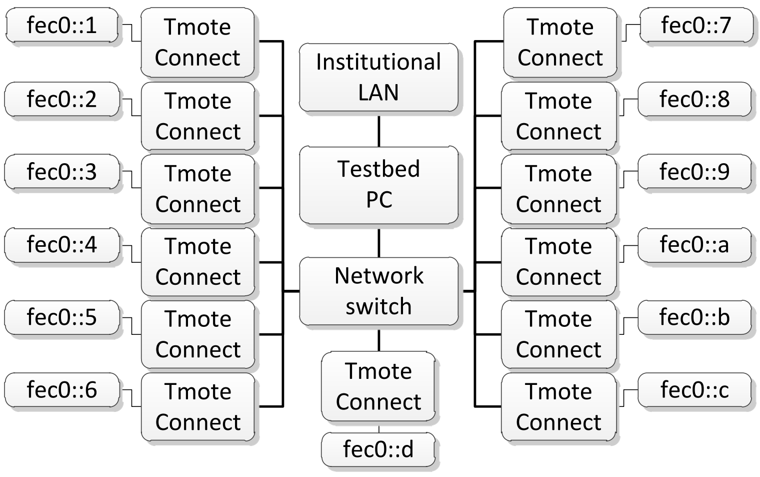


Figure 5.9: Testbed layout

to in Cacti as 'Mote13'. All the PHP generated graphs in this chapter were generated from sensor data from this node. Node 13 was not monitored by Nagios. This was done to minimise the number of ICMP messages sent to the node in order to compare graphs of traffic to and from the node as seen later. Node 13 was programmed with the SNMP enabled `windStation` application which generated UDP messages containing sensor readings. Nodes 1 to 7 were monitored by Nagios and also Cacti but with the focus mainly on Nagios. Nodes 1 to 7 and the remaining nodes were programmed with the `hopNode` application. These nodes would help with routing messages in the network while not generating their own application layer messages.

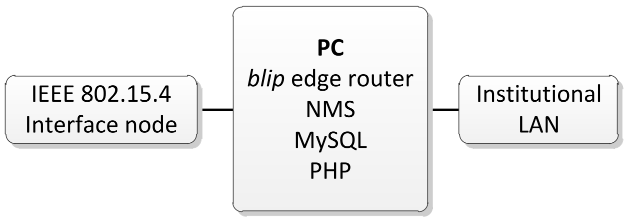


Figure 5.10: Edge router PC

The graphs in Figure 5.11 show data logged from the onboard sensors on WSN node 13 (graphs were from uncalibrated sensor data and values). UDP packets were sent from the `windStation` application to the network monitoring station that used a Python script to log the values to a MySQL database. It was also possible to log SNMP trap messages sent to the network monitoring station in a similar fashion, but this was not fully explored.

The graphs were then created from the data in the database using the JpGraph library

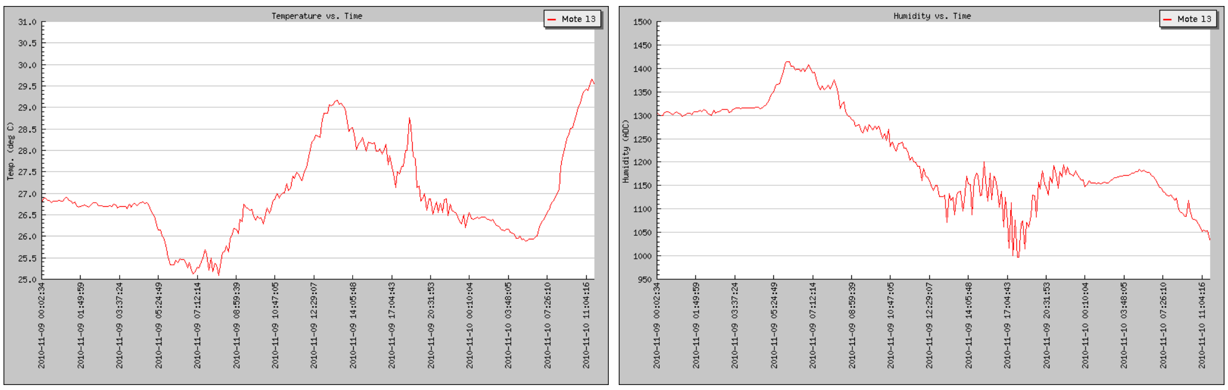


Figure 5.11: PHP Graphs of temperature and humidity as monitored by node 13

and PHP. More information on the graphing and Python logging can be found in the appendices. The sending of sensor data over UDP and graphing of the data was to show that it was still possible to run a sensor network application alongside the SNMPv1 software agent and have the WSN monitored by the SNMP. It also served as a source of traffic to show the functioning of the statistics kept by the `snmpMibP` component on the amount of packets that the main application code sent and received. These statistics were in the form of the `mote.stats.APPsent` and `mote.stats.APPreceived` objects in the MIB.

Figure 5.12 shows another example of sensor data that is graphed using the graphing solution developed. The graph shows readings from one of the light sensors on a node over approximately 36 hours. The first period of daylight (marked *a.*) shows overcast conditions with an increase in the visible light level in the afternoon as more light entered through the west facing windows. The small period of light (marked *b.*) in the evening is due to the lights being switched on in the laboratory for that period. The next day (marked *c.*) shows considerably higher light levels indicating a sunny day with light levels increasing towards the afternoon.

The software package Cacti was used to gather, log and graph data from the WSN. This was done entirely using the SNMPv1 and the software agent implementation described and implemented in this document. Cacti was used to log and graph data from MIB objects over a period of time. This included sensor readings and data describing the flow of messages to and from nodes, i.e. traffic graphs. This also validates the operation of the SNMPv1 agent with other software packages and over a period of time.

Cacti is a PHP driven front end to RRDtool that uses RRDtool to store and graph data. It offers easy templating of graphs and data sources, user management and handles the gathering of data through a poller. Cacti was used to gather data using the SNMP, this data could then be graphed in a variety of different ways. As with Nagios, Cacti is usually used to graph traffic through routers and other network devices on conventional IP networks, as seen in Figure 5.13. With the use of 6LoWPAN and the SNMPv1 software

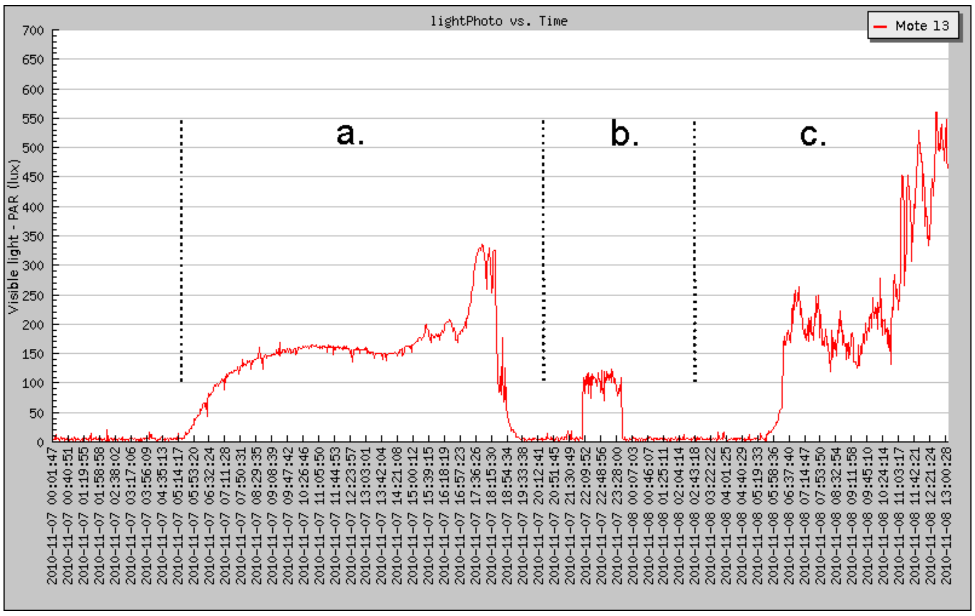


Figure 5.12: PHP graph of captured data from the Photosynthetically-Active Radiation (PAR) light sensor on 'Mote 13'

agent it was possible to use Cacti to graph values that were gathered from the WSN nodes including parameters like temperature and battery voltage, shown in Figure 5.14. It was also possible to graph message traffic through nodes, as will be shown next.

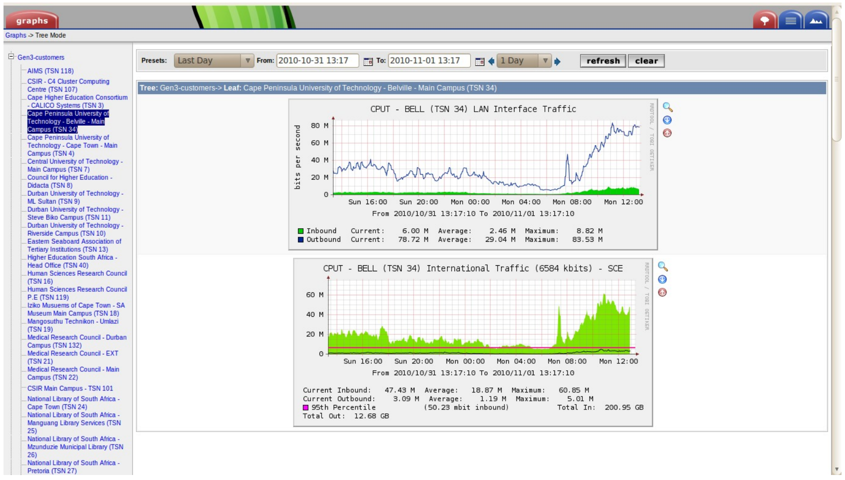

graphs
CPUT - BELL (TSN 34) LAN Interface Traffic
CPUT - BELL (TSN 34) International Traffic (6584 kbits) - SCE
refresh
clear

Figure 5.13: Cacti being used to graph traffic on conventional networks

'RRDtool is the open source industry standard, high performance data logging and graphing system for time series data' (Oetiker, 2009). The Round Robin Database Tool (RRDtool) keeps data at a constant time interval but lets the user update the log file at any moment in time. RRDtool is able to interpolate the value of the data-source at the latest interval and write this value to the log file. RRDtool can store data in several Round Robin Archives (RRA) within a single Round Robin Database (RRD), each archive with its own resolution and consolidation function. Data is written to the RRA in a round robin fashion, as the name suggests, which means that the archive always stays a constant size. (Oetiker, 2011)

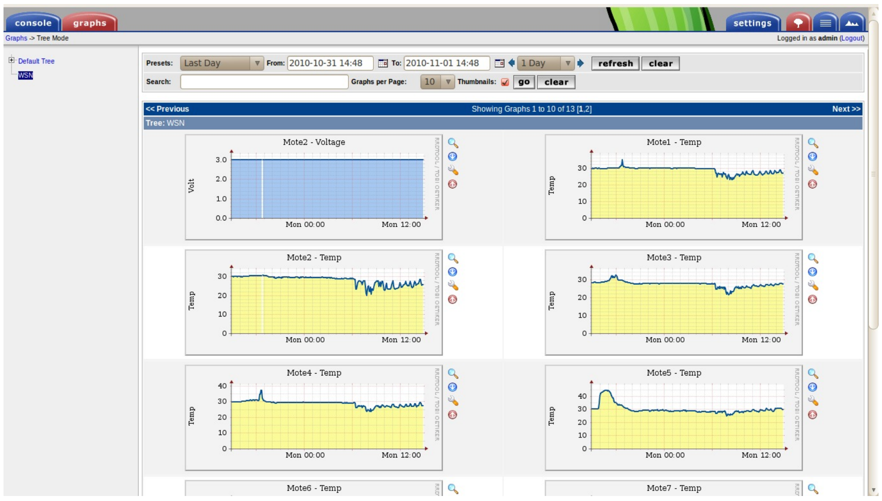


Figure 5.14: Cacti graphing sensor values from WSN nodes

With the use of Cacti and the SNMPv1 software agent it is possible to set up and use a WSN to graph physical sensor readings in a very short period of time with no additional code or software creation needed. If Cacti is already being used to graph other SNMP enabled devices it should be relatively easy to add WSN nodes to the setup.

The traffic generated by the `windStation` WSN application code can be seen in Figure 5.15. The traffic is in the form of UDP messages generated by the node that contains physical sensor readings. The statistic on the number of UDP messages with sensor readings sent by the application (can be seen as application layer messages) is gathered and graphed by Cacti from the `mote.stats.APPsent` object. In this instance the WSN node transmits a packet with sensor readings approximately once every ten minutes. This can be seen in Figure 5.15 with activity at a period of ten minutes.

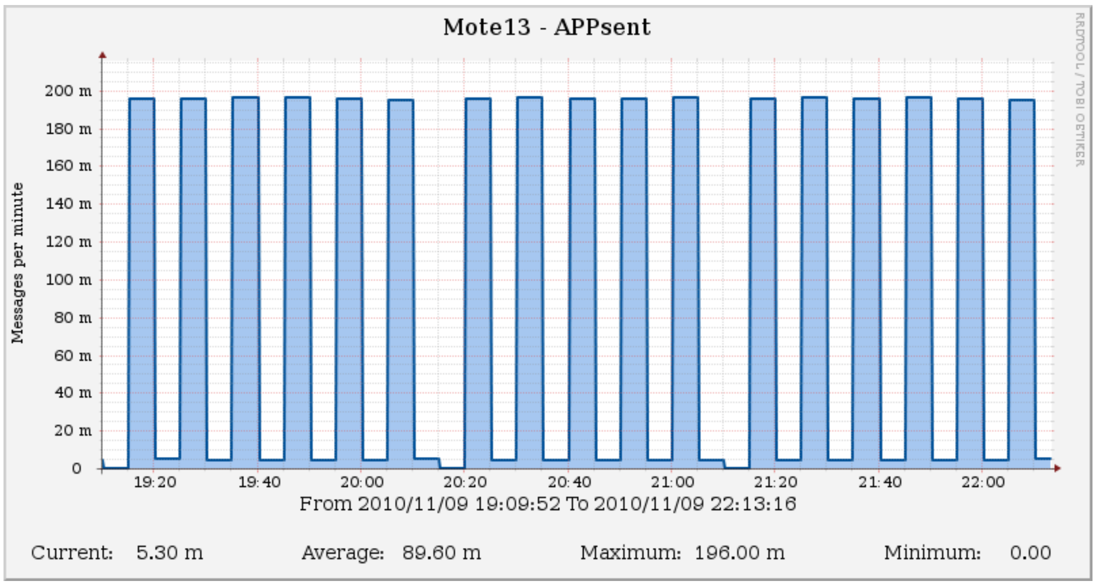


Figure 5.15: Graph of APPsent object data generated by the windStation application

It should be noted that Cacti is currently configured to retrieve information at five minute intervals and also graphs the data five minute intervals at a time. This translates to one message per five minute interval, which is converted to $\frac{1}{5}$ or 0.2 messages per minute, for the periods where there is activity. The Y-axis of the graph has units of messages per minute with the m indicating milli, for example 200 milli messages per minute. Because the node transmits at intervals of ten minutes, there is a five minute gap between the five minute periods of activity. Because the sending of the data packets does not happen at precisely ten minute intervals the point in the five minute interval at which the packet is sent changes with each interval. This causes a gap every few intervals on the graph. This verifies that it is possible to monitor the number of messages a node sends using the developed SNMPv1 agent, with the agent behaving as expected.

Figure 5.16 shows data gathered from the a WSN node's temperature sensor using SNMP and graphed with Cacti. Figure 5.17 shows data from the same sensor over the same time period, but this graph is generated by the created PHP graphing script from data that was sent by the `windStation` application code on the node over UDP and stored in a MySQL database.

From Figure 5.16 the consequences of an unanswered SNMP request from Cacti can be seen. This presents itself as a gap in the graph at approximately 19:45. As can be seen the two graphs created through two different means correlate well and are quite similar. This further indicates the correct operation of the SNMPv1 software agent and its ability to be used with network monitoring software like Cacti.

It is also possible to graph objects from the `stats` section of the MIB, thereby creating traffic graphs of messages sent and received by the WSN node. Figures 5.18, 5.19, 5.20, 5.22 and 5.23 show graphs for the various types of messages that are sent by the WSN node.

Figure 5.18 shows the messages that the application code running on the WSN node generates. These messages contain the sensor readings that are stored and graphed using the PHP graphing scripts. From Figure 5.18 the effects of changing the timer that dictates the generation of the messages can be seen, starting at a high rate the period is increased with another increase at approximately 15:00 on Tuesday. Figure 5.15 shows the area after 15:00 in greater detail. The gap at approximately 10:00 Tuesday is most likely due to unanswered SNMP requests. The ICMP messages that the WSN node sends can be seen in Figure 5.19. As can be seen from both these graphs the amount of traffic generated is relatively low for the period shown.

The SNMP messages sent by the WSN node can be seen in Figure 5.20, while Figure 5.21 shows the data gathered for the `SNMPreceived` object. The graphs for sent and received SNMP messages should be similar because the SNMP messages are sent only when requested, provided no errors occurred in the requesting of the data via SNMP. Errors are possible because the SNMP makes use of the unreliable UDP transport service.

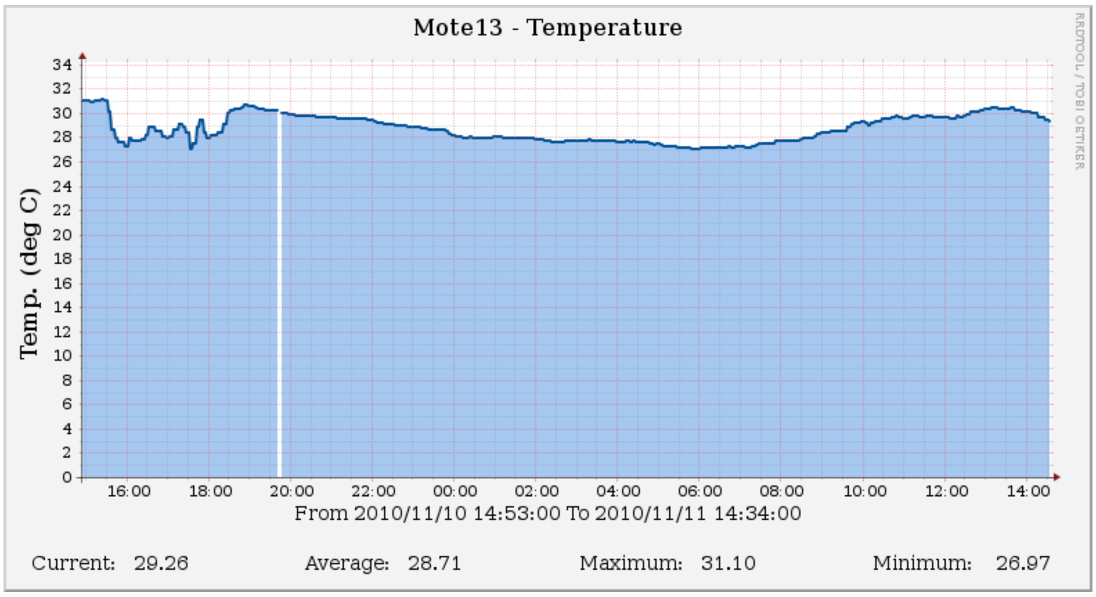


Figure 5.16: Temperature data gathered over SNMP with Cacti

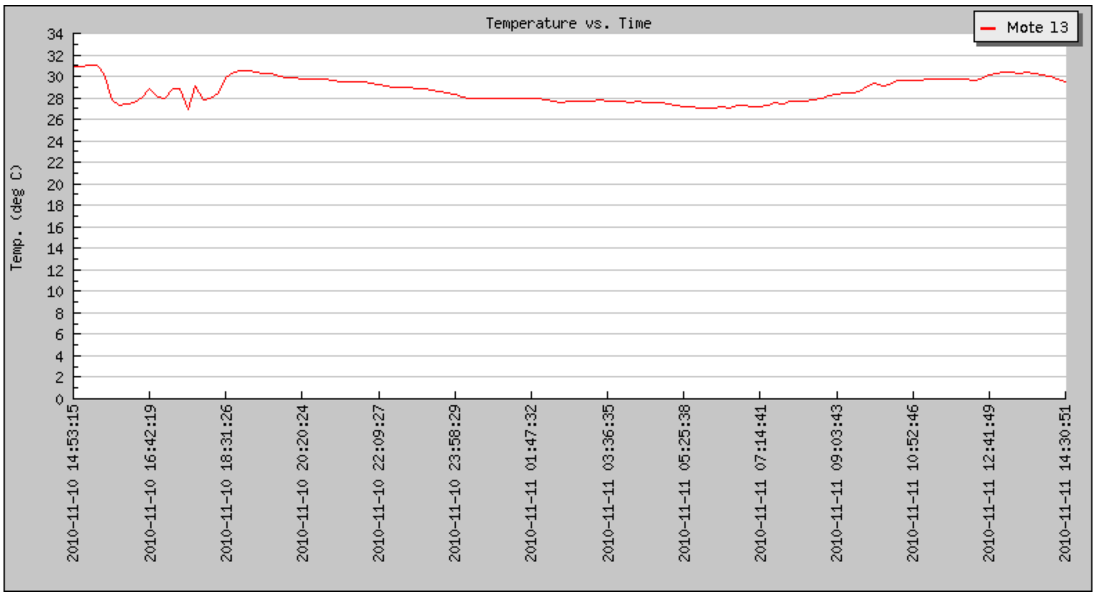


Figure 5.17: Temperature data from application code sent over UDP and graphed using PHP

The UDP messages sent from the WSN node can be seen in Figure 5.22. UDP is the only transport layer protocol that is used on the WSN node. The graphs for sent UDP and IP messages should therefore be similar. It should however be noted that the sent ICMP messages are not counted under UDP sent messages but are counted under IP sent messages. The sent UDP messages in Figure 5.22 are a combination of the sent SNMP messages (Figure 5.20) and the messages sent by the WSN application code running on the node (Figure 5.18). This can be seen on early Tuesday morning at around 01:00 when the graphs are steady with no big spikes. The ripple from the APPsent graph (Figure

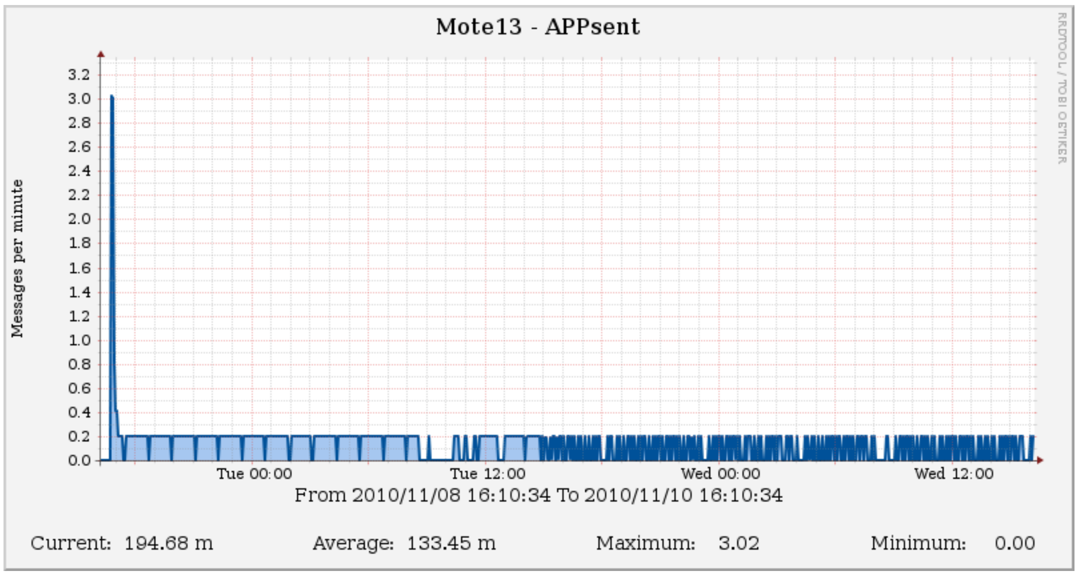


Figure 5.18: Cacti graph of the APPsent object in MIB

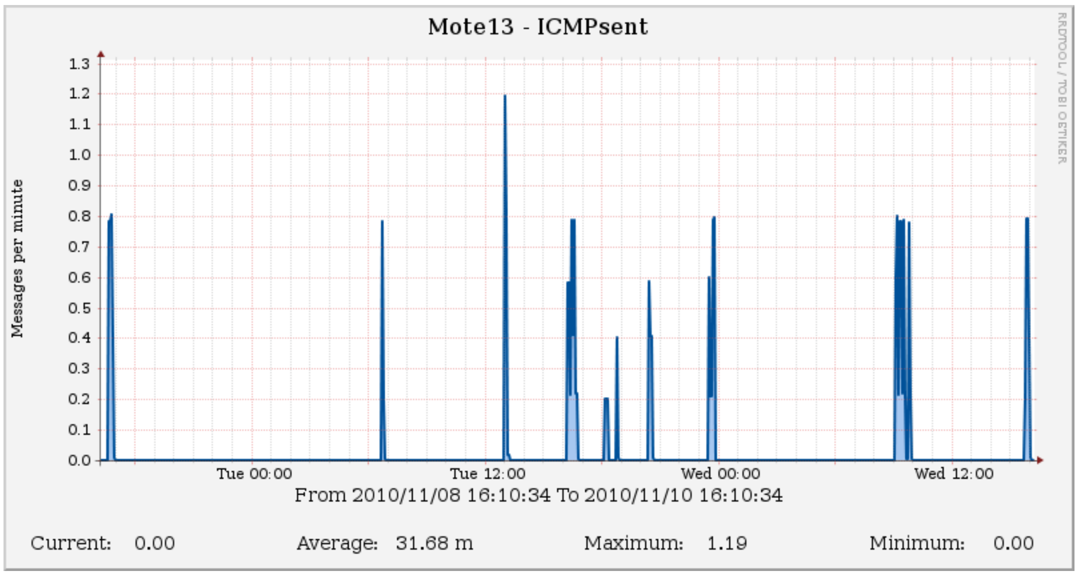


Figure 5.19: Cacti graph of ICMP messages sent by node

5.18) can be seen in the UDPsent graph (Figure 5.22) including an increase of about 0.2 messages a minute in the average level.

Figure 5.23 shows the IP messages sent by the node. This graph indicates all the messages sent by the node because IP is the only network layer protocol being used. The IP sent messages graph (Fig. 5.23) is therefore a combination of ICMP (Fig. 5.19) and UDP (Fig. 5.22) sent messages. Because the amount of ICMP messages sent is low it is difficult to see their effect on the IP sent messages graph.

The cause for the increase in traffic between Tuesday 23:00 and Wednesday 09:00 is unknown (Figures 5.20, 5.21, 5.22, 5.23). It is unlikely that it is caused by manual sending of SNMP requests as it happened during the night and early morning. But as

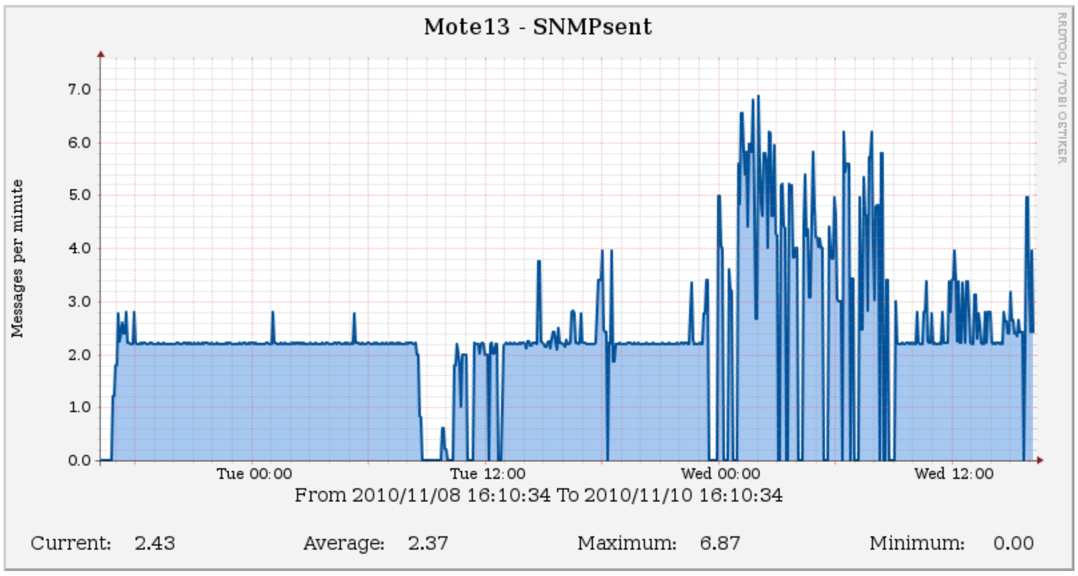


Figure 5.20: Graph generated by Cacti of SNMP messages sent by WSN node

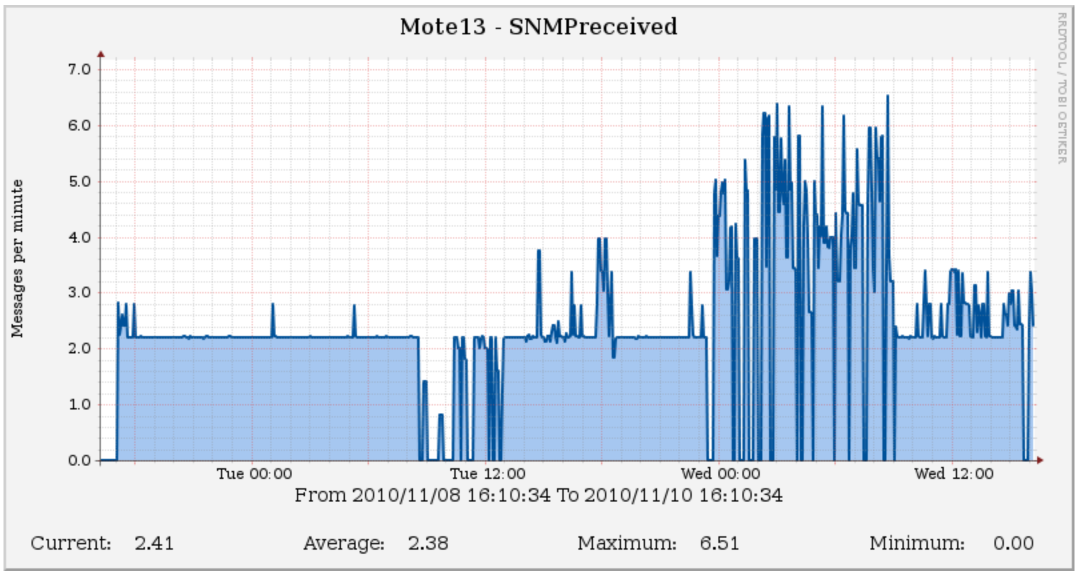


Figure 5.21: Graph generated by Cacti of SNMP messages received by WSN node

can be seen from the graphs it is most likely requests coming from Cacti as it is also present in the received SNMP messages graph. This can be caused by Cacti not receiving replies from the node and then re-sending requests with both the original request and additional requests being logged in the statistics of the node.

The messages that a WSN node forwards for other nodes can also be monitored using the `stats.IPforwarded` object. Figures 5.24 and 5.25 shows the messages that two nodes forwarded for their neighbours.

The routing of messages through the network changes regularly and also changes according to nodes joining and leaving the network. The amount of messages a node forwards for other nodes is relevant because this affects the amount of battery power a

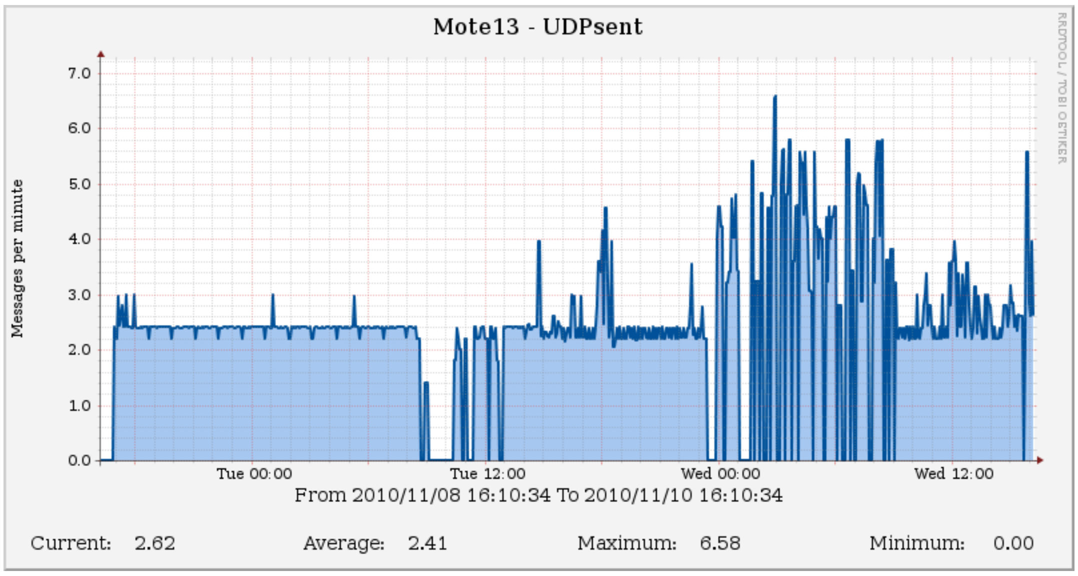


Figure 5.22: Cacti graph of UDP packets sent by node

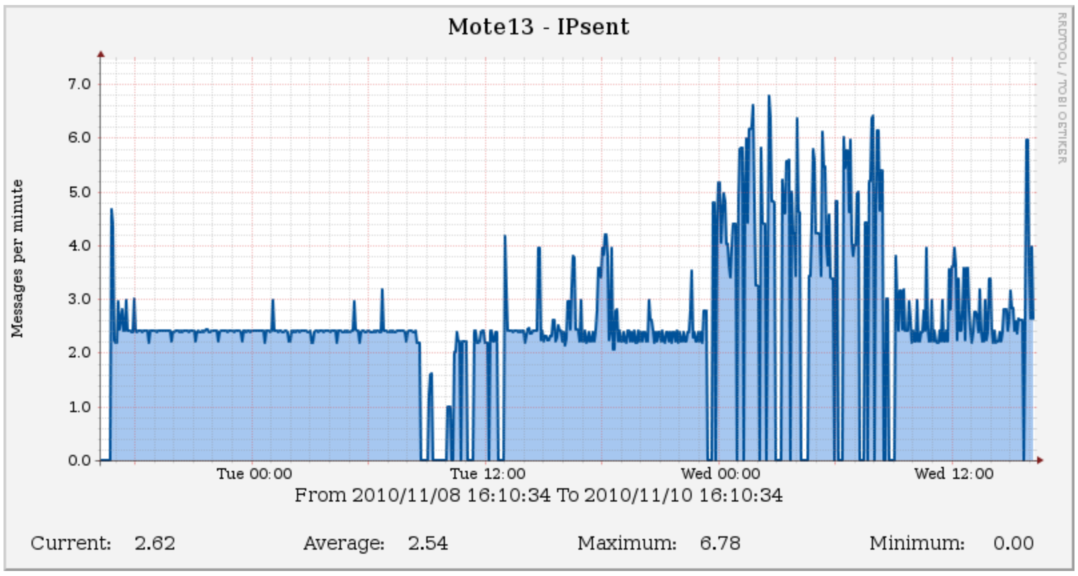


Figure 5.23: Cacti graph of IP packets sent by the WSN node

node consumes.

The second network monitoring software package used was Nagios. Nagios does not offer graphing of the data it captures and by default monitors nodes using ICMP echo requests. It is possible to extend the functionality of Nagios with the addition of plugins and addons. Adding monitoring of nodes by using the SNMP to Nagios is possible and two examples can be seen in the following paragraphs. Nagios can generate availability reports for hosts and services and it is also possible to view event logs. This can be helpful when diagnosing problems or to determine the time at which a specific node failed.

Monitoring of various services can be set up in Nagios. When the Nagios package is installed the default monitoring is that of the `PING` (ICMP ECHO_REQUEST) service.

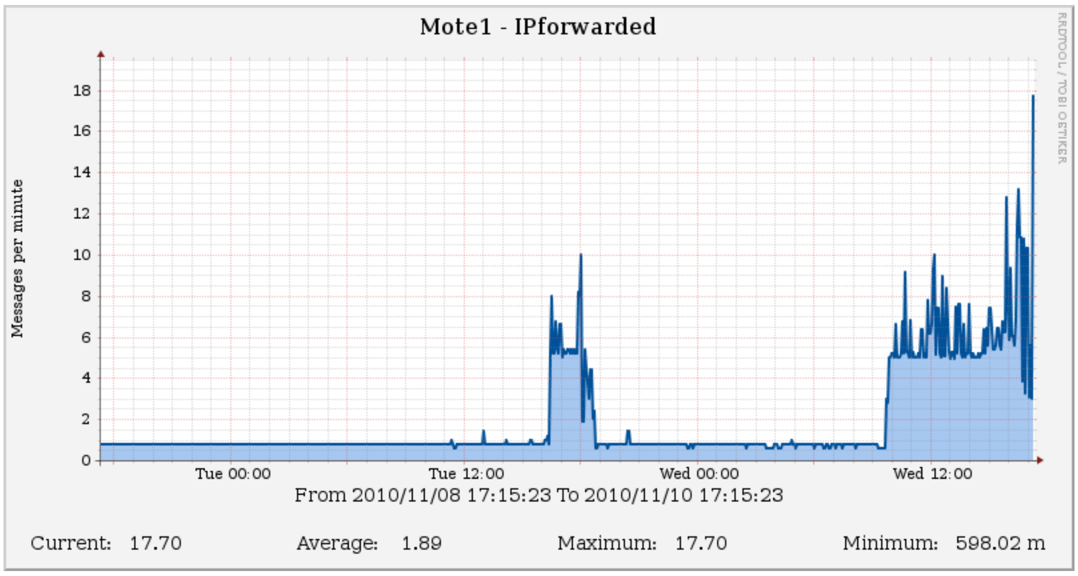


Figure 5.24: Cacti graph of IP packets forwarded by node 1

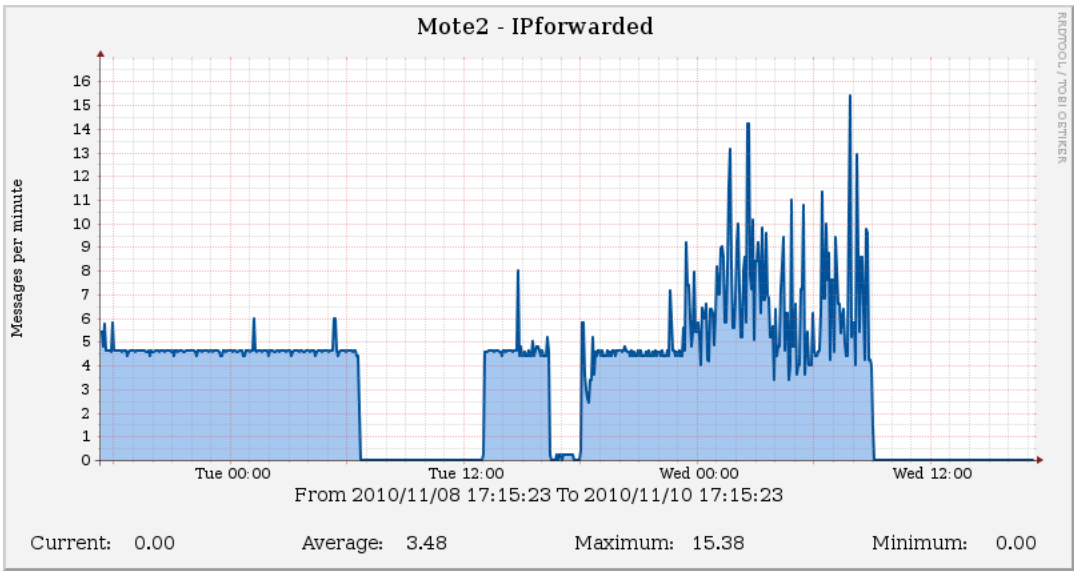
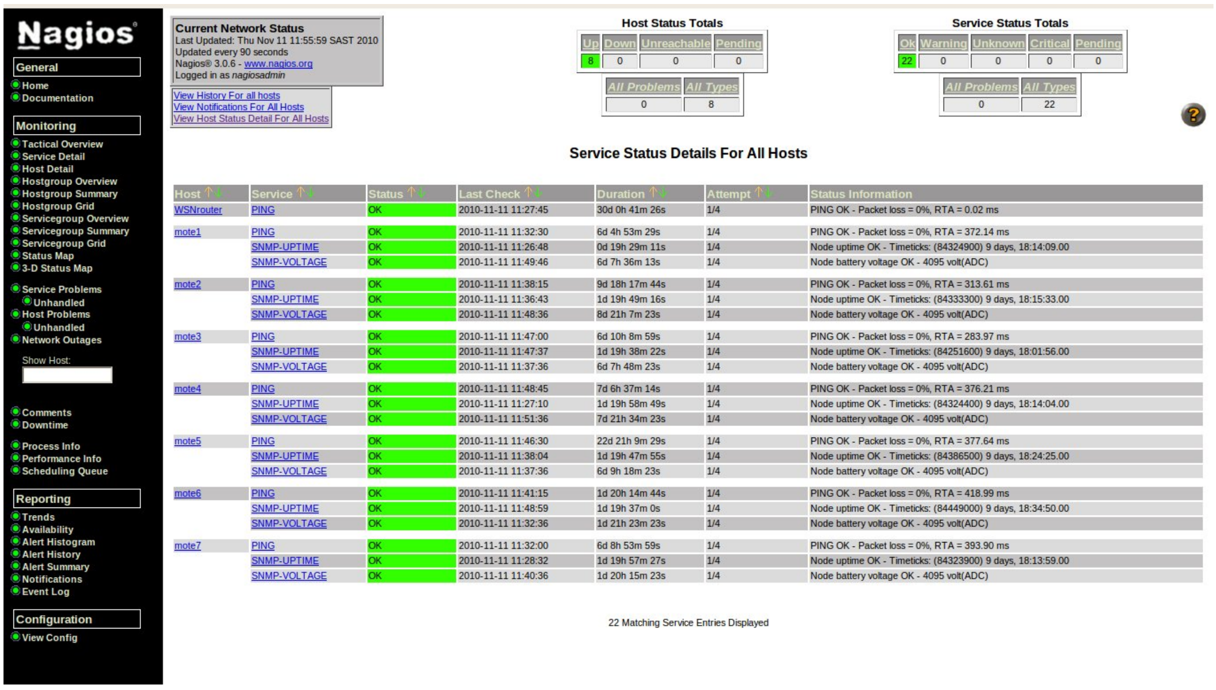


Figure 5.25: Cacti graph of IP packets forwarded by node 2

In addition to this, two other parameters were added and monitored using the SNMP. The services `SNMP-UPTIME` and `SNMP-VOLTAGE` were created. These monitored a node's battery voltage and uptime counter. The battery voltage is given as a raw uncalibrated ADC value of 4095.

For each service a level can be set at which a warning is given or a critical warning is given. Nagios can then be configured to send notifications when these states occur. An appropriate value for the node battery voltage can be set and a warning is then generated when the returned node voltage falls below the specified value. No such warning values were set for the monitoring of the node uptime. The monitoring of node uptime was done to check nodes for unexpected resets.

Figure 5.26 shows the ‘Service Detail’ page that contains the current network status. From this page the status of each service being monitored on each node can be seen. The page shows the state of the service and the duration of that current state. It also shows information on the status of each service. For the `PING` service the packet loss and round trip average (RTA) is shown and the value for the uptime and voltage is shown for `SNMP-UPTIME` and `SNMP-VOLTAGE`.



Current Network Status
Last Updated: Thu Nov 11 11:55:59 SAST 2010
Updated every 90 seconds
Nagios® 3.0.6 - www.nagios.org
Logged in as *nagiosadmin*

Host Status Totals

Up	Down	Unreachable	Pending
8	0	0	0

All Problems	All Types
0	8

Service Status Totals

Ok	Warning	Unknown	Critical	Pending
22	0	0	0	0

All Problems	All Types
0	22

Service Status Details For All Hosts

Host	Service	Status	Last Check	Duration	Attempt	Status Information
WSNrouter	PING	OK	2010-11-11 11:27:45	30d 0h 41m 26s	1/4	PING OK - Packet loss = 0%, RTA = 0.02 ms
mote1	PING	OK	2010-11-11 11:32:30	6d 4h 53m 29s	1/4	PING OK - Packet loss = 0%, RTA = 372.14 ms
	SNMP-UPTIME	OK	2010-11-11 11:26:48	0d 19h 29m 11s	1/4	Node uptime OK - Timeticks: (84324900) 9 days, 18:14:09.00
	SNMP-VOLTAGE	OK	2010-11-11 11:49:46	6d 7h 36m 13s	1/4	Node battery voltage OK - 4095 volt(ADC)
mote2	PING	OK	2010-11-11 11:38:15	9d 18h 17m 44s	1/4	PING OK - Packet loss = 0%, RTA = 313.61 ms
	SNMP-UPTIME	OK	2010-11-11 11:36:43	1d 19h 49m 16s	1/4	Node uptime OK - Timeticks: (84333300) 9 days, 18:15:33.00
	SNMP-VOLTAGE	OK	2010-11-11 11:48:36	8d 21h 7m 23s	1/4	Node battery voltage OK - 4095 volt(ADC)
mote3	PING	OK	2010-11-11 11:47:00	6d 10h 8m 59s	1/4	PING OK - Packet loss = 0%, RTA = 283.97 ms
	SNMP-UPTIME	OK	2010-11-11 11:47:37	1d 19h 38m 22s	1/4	Node uptime OK - Timeticks: (84251600) 9 days, 18:01:56.00
	SNMP-VOLTAGE	OK	2010-11-11 11:37:36	6d 7h 48m 23s	1/4	Node battery voltage OK - 4095 volt(ADC)
mote4	PING	OK	2010-11-11 11:48:45	7d 6h 37m 14s	1/4	PING OK - Packet loss = 0%, RTA = 376.21 ms
	SNMP-UPTIME	OK	2010-11-11 11:27:10	1d 19h 58m 49s	1/4	Node uptime OK - Timeticks: (84324400) 9 days, 18:14:04.00
	SNMP-VOLTAGE	OK	2010-11-11 11:51:36	7d 21h 34m 23s	1/4	Node battery voltage OK - 4095 volt(ADC)
mote5	PING	OK	2010-11-11 11:46:30	22d 21h 9m 29s	1/4	PING OK - Packet loss = 0%, RTA = 377.64 ms
	SNMP-UPTIME	OK	2010-11-11 11:38:04	1d 19h 47m 55s	1/4	Node uptime OK - Timeticks: (84386500) 9 days, 18:24:25.00
	SNMP-VOLTAGE	OK	2010-11-11 11:37:36	6d 9h 18m 23s	1/4	Node battery voltage OK - 4095 volt(ADC)
mote6	PING	OK	2010-11-11 11:41:15	1d 20h 14m 44s	1/4	PING OK - Packet loss = 0%, RTA = 418.99 ms
	SNMP-UPTIME	OK	2010-11-11 11:48:59	1d 19h 37m 0s	1/4	Node uptime OK - Timeticks: (84449000) 9 days, 18:34:50.00
	SNMP-VOLTAGE	OK	2010-11-11 11:32:36	1d 21h 23m 23s	1/4	Node battery voltage OK - 4095 volt(ADC)
mote7	PING	OK	2010-11-11 11:32:00	6d 8h 53m 59s	1/4	PING OK - Packet loss = 0%, RTA = 393.90 ms
	SNMP-UPTIME	OK	2010-11-11 11:28:32	1d 19h 57m 27s	1/4	Node uptime OK - Timeticks: (84323900) 9 days, 18:13:59.00
	SNMP-VOLTAGE	OK	2010-11-11 11:40:36	1d 20h 15m 23s	1/4	Node battery voltage OK - 4095 volt(ADC)

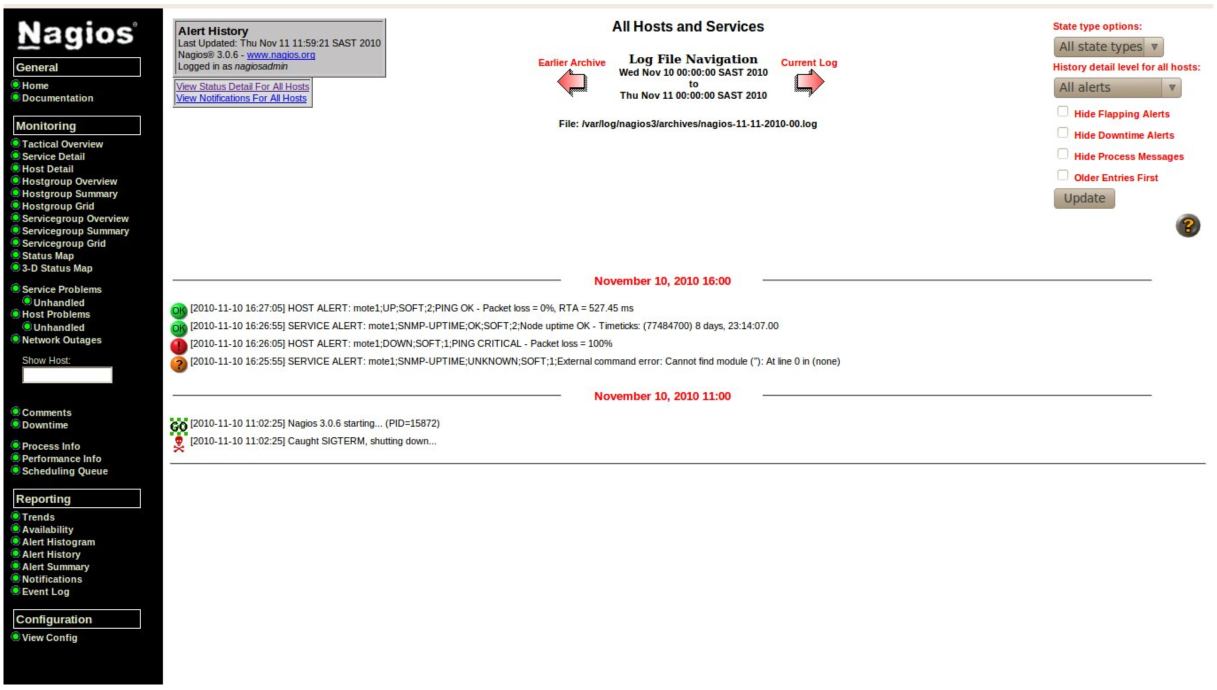
22 Matching Service Entries Displayed

Figure 5.26: Nagios Service Detail page

An alert history for all hosts can also be viewed. The alert history page can be seen in Figure 5.27. This is a log of all events and status changes of services. At 11:00 the history indicates that Nagios was restarted. At 16:00 an indication is given that an unknown problem occurred with checking the SNMP-UPTIME service for mote1. This is followed by an alert that the host is down because of a 100% packet loss checking the PING service. After a short period of time the checks are run again, this time the node replies with an uptime and also responds to a PING service check.

Nagios is also able to generate reports of trends, and availability reports for each node and service. Figure 5.28 shows a report on trends for the ‘SNMP-VOLTAGE’ on ‘mote6’ while Figure 5.29 shows an availability report for the same node and service.

Both Cacti and Nagios have endless configuration options and to master and utilise these packages to their full extent would take a considerable amount of time. The configurations shown for these packages show what is possible with a limited amount of time available. It is, for example, possible to graph multiple values on a single graph with Cacti and to then add other statistics like 95th percentile and total values to the graph.



Alert History
Last Updated: Thu Nov 11 11:59:21 SAST 2010
Nagios® 3.0.6 - www.nagios.org
Logged in as *nagiosadmin*

View Status Detail For All Hosts
View Notifications For All Hosts

All Hosts and Services

Log File Navigation
Wed Nov 10 00:00:00 SAST 2010
to
Thu Nov 11 00:00:00 SAST 2010

File: /var/log/nagios3/archives/nagios-11-11-2010-00.log

State type options: All state types
History detail level for all hosts: All alerts
[ ] Hide Flapping Alerts
[ ] Hide Downtime Alerts
[ ] Hide Process Messages
[ ] Older Entries First

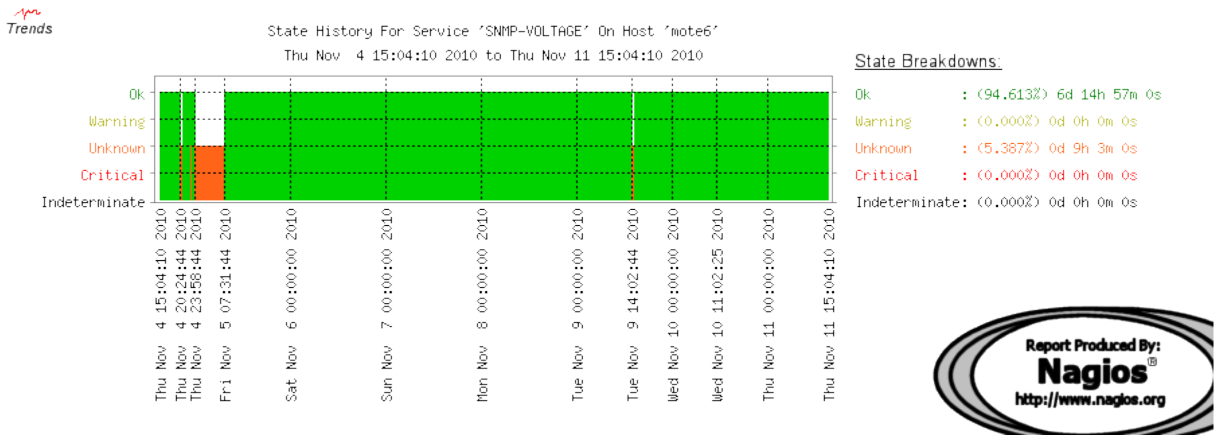
November 10, 2010 16:00

```
[2010-11-10 16:27:05] HOST ALERT: mote1;UP;SOFT;2;PING OK - Packet loss = 0%, RTA = 527.45 ms
[2010-11-10 16:26:55] SERVICE ALERT: mote1;SNMP-UPTIME;OK;SOFT;2;Node uptime OK - Timeticks: (77484700) 8 days, 23:14:07.00
[2010-11-10 16:26:05] HOST ALERT: mote1;DOWN;SOFT;1;PING CRITICAL - Packet loss = 100%
[2010-11-10 16:25:55] SERVICE ALERT: mote1;SNMP-UPTIME;UNKNOWN;SOFT;1;External command error: Cannot find module (''): At line 0 in (none)
```

November 10, 2010 11:00

```
[2010-11-10 11:02:25] Nagios 3.0.6 starting... (PID=15872)
[2010-11-10 11:02:25] Caught SIGTERM, shutting down...
```

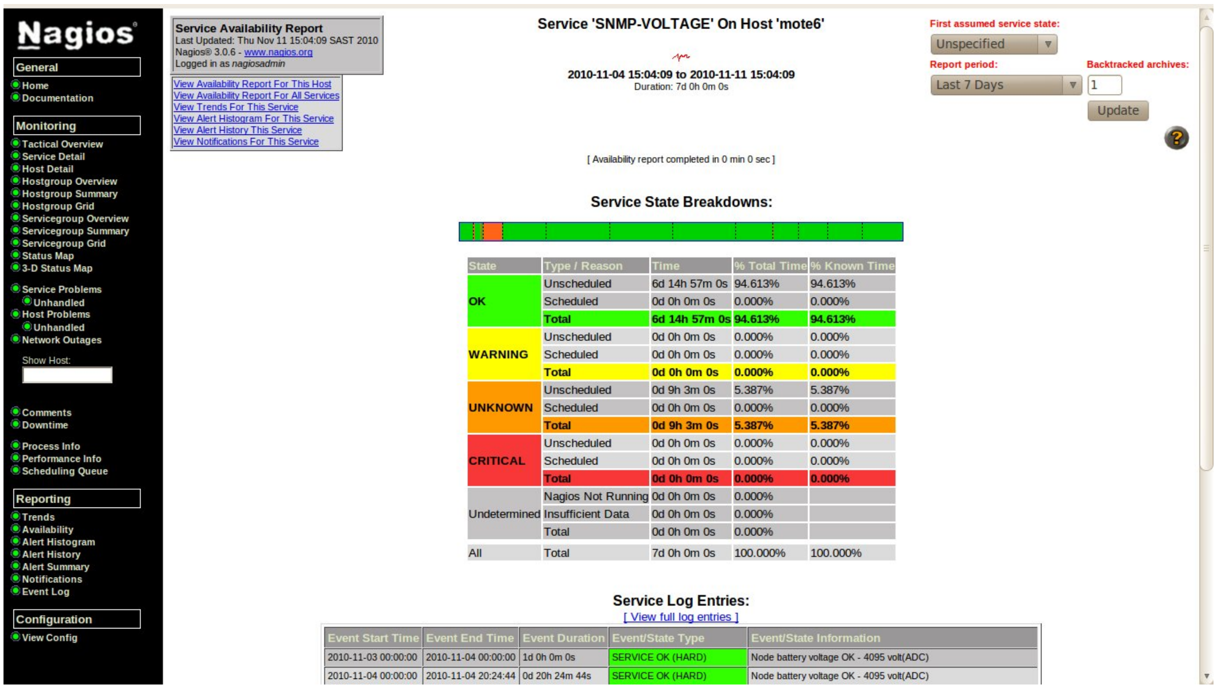
Figure 5.27: Nagios History for all hosts page



State Breakdowns:

```
Ok           : (94.613%) 6d 14h 57m 0s
Warning      : (0.000%) 0d 0h 0m 0s
Unknown      : (5.387%) 0d 9h 3m 0s
Critical     : (0.000%) 0d 0h 0m 0s
Indeterminate: (0.000%) 0d 0h 0m 0s
```

Figure 5.28: Nagios Trends report



Service Availability Report
Last Updated: Thu Nov 11 15:04:09 SAST 2010
Nagios® 3.0.6 - www.nagios.org
Logged in as *nagiosadmin*

Service 'SNMP-VOLTAGE' On Host 'mote6'

2010-11-04 15:04:09 to 2010-11-11 15:04:09
Duration: 7d 0h 0m 0s

First assumed service state: Unspecified
Report period: Last 7 Days
Backtracked archives: 1

[Availability report completed in 0 min 0 sec]

Service State Breakdowns:

State	Type / Reason	Time	% Total Time	% Known Time
OK	Unscheduled	6d 14h 57m 0s	94.613%	94.613%
	Scheduled	0d 0h 0m 0s	0.000%	0.000%
	Total	**6d 14h 57m 0s**	**94.613%**	**94.613%**
WARNING	Unscheduled	0d 0h 0m 0s	0.000%	0.000%
	Scheduled	0d 0h 0m 0s	0.000%	0.000%
	Total	**0d 0h 0m 0s**	**0.000%**	**0.000%**
UNKNOWN	Unscheduled	0d 9h 3m 0s	5.387%	5.387%
	Scheduled	0d 0h 0m 0s	0.000%	0.000%
	Total	**0d 9h 3m 0s**	**5.387%**	**5.387%**
CRITICAL	Unscheduled	0d 0h 0m 0s	0.000%	0.000%
	Scheduled	0d 0h 0m 0s	0.000%	0.000%
	Total	**0d 0h 0m 0s**	**0.000%**	**0.000%**
Undetermined	Nagios Not Running	0d 0h 0m 0s	0.000%	
	Insufficient Data	0d 0h 0m 0s	0.000%	
	Total	0d 0h 0m 0s	0.000%	
All	Total	7d 0h 0m 0s	100.000%	100.000%

Service Log Entries:
[View full log entries]

Event Start Time	Event End Time	Event Duration	Event/State Type	Event/State Information
2010-11-03 00:00:00	2010-11-04 00:00:00	1d 0h 0m 0s	SERVICE OK (HARD)	Node battery voltage OK - 4095 volt(ADC)
2010-11-04 00:00:00	2010-11-04 20:24:44	0d 20h 24m 44s	SERVICE OK (HARD)	Node battery voltage OK - 4095 volt(ADC)

Figure 5.29: Nagios Availability report

From this chapter it can be seen that the SNMPv1 software implementation behaves as expected with each of the software packages it was tested with. It is also possible to gather information relevant to the operation of the node the agent is running on. The correctness of operation was illustrated with commands available in the Net-SNMP package, Nagios and Cacti. Nagios and Cacti also show the reliable operation of the SNMP software agent over an extended period of time using standard network monitoring tools.

Chapter 6

Conclusion

In the introductory chapter the methodology and significance was stated along with an introduction to - and the objectives of the research.

A general introduction to wireless sensor networks was given and also a general background on the different areas of the research. This included the way in which communication between networked nodes is divided into different layers, information on the relevant lower network layers and also the Simple Network Management Protocol. Version six of the Internet Protocol was briefly discussed along with 6LoWPAN and the IEEE 802.15.4 standard. The three versions of the SNMP were examined along with common concepts used in the protocol. The philosophy and thinking behind the implementation of the SNMP agent was discussed and also WSN specific monitoring and similar work.

The arrival of 6LoWPAN and 6LoWPAN implementations have made it easier to start looking at existing protocols, techniques and software and apply them as is to WSNs. One of these protocols is the Simple Network Management Protocol (SNMP) and was further explored in this work. After some consideration the decision was made to implement as much of the first version of the SNMP as possible. This implementation can then be used to evaluate the basic functionality that using SNMP would bring to LoWPANs.

The structure of an SNMP GetRequest message was examined and discussed in more detail giving a better understanding of the structure of SNMP messages in general. The implementation of the SNMPv1 software agent was discussed with some code snippets, wiring diagrams and general flow diagrams for each component shown. The division of the SNMP software agent into different components was discussed and each of the TinyOS components created were examined in more detail. The demonstration WSN applications created were also discussed with the aid of wiring diagrams showing the connected components and interfaces used.

The agent implementation was then tested and evaluated to show correctness of operation and usability. Testing was divided into 'workstation' and 'testbed' testing. For the workstation testing the agent was queried using Net-SNMP and the output from the commands was used to check for correctness. The responses were also evaluated at packet level using Wireshark. In the testbed section the SNMP agent was installed on a WSN testbed with multiple nodes and it was possible to show its reliable operation over time using standard network monitoring tools.

The SNMPv1 agent implementation also offers an easy way to set up a WSN without the need to do extensive coding. If the user is familiar with packages like Cacti and Nagios it is easy to do monitoring and set up fairly complex graphs of parameters within the network and also of node sensor values. It also now becomes easy to integrate a WSN into existing network monitoring infrastructure.

This starts to touch on the assumed benefits that Kushalnagar *et al.* (2007) mentions in RFC 4919, one of which is the reuse of existing infrastructure. This is done by using existing networks to connect the WSN and also incorporating the WSN into existing management and monitoring infrastructure. The SNMP agent implementation also shows that it is possible to reuse existing protocols and software on WSNs by making use of 6LoWPAN.

The objective of implementing and testing an SNMPv1 agent in TinyOS was achieved along with showing the basic functionality it brings to LoWPANs. The SNMP could be used without the use of translation gateways or proxies for the SNMP.

Existing software and tools like Cacti and Wireshark could also be reused. With the use of the SNMP, WSNs can be easily integrated into existing network monitoring activities.

This also starts to confirm some of the assumed benefits of 6LoWPAN, like tool reuse. The created software agent can easily be used in future WSN deployments by the Centre for Instrumentation Research (CIR), while the constructed WSN testbed can be used in future research. Wireless sensor network monitoring using the Simple Network Management Protocol was also achieved.

6.1 Evaluation of work done

One of the texts examined for this research stated that although the protocol is called the 'Simple' Network Management Protocol, there is nothing trivial about it. Even the first version of the protocol presents a steep learning curve with multiple concepts to be grasped and having ASN.1 and BER to contend with.

As can be seen from the last subsection in section 4.4, code size and its link to complexity is an issue. Even the much lighter weight first version of the SNMP along with the 6LoWPAN stack consumes most of the resources available on the node. Therefore, there is not much hope that a full-featured implementation of the third version would be feasible on the platform used.

The use of request and response type information gathering on WSNs may not be the most energy efficient way of gathering information. Trap messages could therefore be used to greater effect.

6.2 Proposed future work and recommendations

Although it was shown that it is possible to use the SNMP as is on WSNs, a more efficient solution would be to optimise the protocol for use on LoWPANs. It would be possible to compress some of the SNMP headers or remove headers and features that might not be needed in WSNs. An example is the `error-status` and `error-index` fields which could be compressed because most of the time these fields indicate a no error state. The security needs for the SNMP should also be evaluated, this evaluation should take into account all the network layers including the 6LoWPAN adaptation layer. It might be possible to consolidate security into a single security layer, which is used by other protocols and not just the SNMP, possibly below the Application layer.

It might also be advantageous to start looking at more able WSN hardware platforms with more resources available to power the next generation of IP enabled WSN nodes.

Greater use of SNMP Trap messages should be considered, where the generation of the Trap messages could be controlled with SNMP SetRequests. This would almost certainly halve the number of messages generated by the current configuration and reduce energy consumption. The current agent implementation and the network monitoring software configuration can be optimised for greater energy efficiency. Optimisations can be done to reduce the size of the application code generated, this could depend largely on the specific application.

References

Archrock, 2010, *Arch Rock*, URL `http://www.archrock.com/`, Last visited [April 2010].

Astrin, A., 2011, *IEEE 802.15 WPAN Task Group 6 (TG6) Body Area Networks*, URL `http://www.ieee802.org/15/pub/TG6.html`, Last visited [10 August 2011].

Baker, F., 1995, *Requirements for IP Version 4 Routers*, RFC 1812, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc1812.txt`, Last visited [6 July 2011].

Barr, J., 2009, *IEEE 802.15 WPAN Task Group 3 (TG3)*, URL `http://www.ieee802.org/15/pub/TG3.html`, Last visited [10 August 2011].

Blumenthal, U. & Wijnen, B., 2002, *User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)*, RFC 3414, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc3414.txt`, Last visited [7 November 2010].

Braden, R., 1989, *Requirements for Internet Hosts – Communication Layers*, RFC 1122, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc1122.txt`, Last visited [7 November 2010].

Cacti, 2010, *The Complete RRDTool-based Graphing Solution*, URL `http://www.cacti.net/`, Last visited [6 November 2010].

Case, J., Fedor, M., Schoffstall, M. & Davin, J., 1990, *A Simple Network Management Protocol (SNMP)*, RFC 1157, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc1157.txt`, Last visited [7 November 2010].

Case, J., McCloghrie, K., Rose, M. & Waldbusser, S., 1993, *Introduction to version 2 of the Internet-standard Network Management Framework*, RFC 1441, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc1441.txt`, Last visited [7 November 2010].

Case, J., McCloghrie, K., Rose, M. & Waldbusser, S., 1996a, *Introduction to Community-based SNMPv2*, RFC 1901, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc1901.txt`, Last visited [7 November 2010].

Case, J., McCloghrie, K., Rose, M. & Waldbusser, S., 1996b, *Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)*, RFC 1905, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc1905.txt`, Last visited [7 November 2010].

Case, J., Mundy, R., Partain, D. & Stewart, B., 2002, *Introduction and Applicability Statements for Internet Standard Management Framework*, RFC 3410, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc3410.txt`, Last visited [7 November 2010].

Cerf, V., 1988, *IAB Recommendations for the Development of Internet Network Management Standards*, RFC 1052, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc1052.txt`, Last visited [7 November 2010].

Cerf, V., 1989, *Report of the Second Ad Hoc Network Management Review Group*, RFC 1109, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc1109.txt`, Last visited [7 November 2010].

Choi, H., Kim, N. & Cha, H., 2009, *6LoWPAN-SNMP: Simple Network Management Protocol for 6LoWPAN*, in *High Performance Computing and Communications, 10th IEEE International Conference on*, 305–313, URL `http://www.computer.org/portal/web/csdl/doi/10.1109/HPCC.2009.49`, Last visited [6 November 2010].

Cisco, 2010a, *Cisco Has Acquired Arch Rock*, URL `http://www.archrock.com/`, Last visited [12 October 2011].

Cisco, 2010b, *Internetworking Technology Handbook - Simple Network Management Protocol (SNMP) - Cisco Systems*, URL `http://www.cisco.com/en/US/docs/internetworking/technology/handbook/SNMP.html`, Last visited [6 November 2010].

Crossbow, 2011, *Moob Crossbow*, URL `http://www.xbow.com`, Last visited [12 October 2011].

Dawson-Haggerty, S. & Tavakoli, A., 2010, *Berkeley IP Information*, URL `http://smote.cs.berkeley.edu:8000/tracenv/wiki/blip`, Last visited [31 October 2010].

Deering, S. & Hinden, R., 1998, *Internet Protocol, Version 6 (IPv6) Specification*, RFC 2460, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc2460.txt`, Last visited [7 November 2010].

Dunkels, A., 2010, *The Contiki Operating System*, URL `http://www.sics.se/contiki/`, Last visited [6 November 2010].

Frye, R., Levi, D., Routhier, S. & Wijnen, B., 2000, *Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework*, RFC 2576, Internet Engineering Task Force, URL http://www.rfc-editor.org/rfc/rfc2576.txt, Last visited [7 November 2010].

Galvin, J. & McCloghrie, K., 1993a, *Administrative Model for version 2 of the Simple Network Management Protocol (SNMPv2)*, RFC 1445, Internet Engineering Task Force, URL http://www.rfc-editor.org/rfc/rfc1445.txt, Last visited [7 November 2010].

Galvin, J. & McCloghrie, K., 1993b, *Security Protocols for version 2 of the Simple Network Management Protocol (SNMPv2)*, RFC 1446, Internet Engineering Task Force, URL http://www.rfc-editor.org/rfc/rfc1446.txt, Last visited [7 November 2010].

Gay, D., Levis, P., Culler, D. & Brewer, E., 2003, *nesC 1.1 Language Reference Manual*, URL http://nescc.sourceforge.net/papers/nesc-ref.pdf, Last visited [6 November 2010].

Gifford, I., 2004, *IEEE 802.15 WPAN Task Group 1 (TG1)*, URL http://www.ieee802.org/15/pub/TG1.html, Last visited [10 August 2011].

Han, R., 2007, *MANTIS : HomePage*, URL http://mantisos.org/index/tiki-index.php.html, Last visited [12 October 2011].

Harvan, M., 2007, *A 6LoWPAN implementation for TinyOS 2.x*, URL http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x/tos/lib/net/6lowpan/README?view=markup, Last visited [6 November 2010], can also be found at $TOSROOT/tos/lib/net/6lowpan/README.

Heile, B., 2011, *IEEE 802.15 Working Group for WPAN*, URL http://www.ieee802.org/15/, Last visited [10 August 2011].

Hinden, R.M., 1996, *IP next generation overview*, Communications of the ACM, 39:61–71, URL http://doi.acm.org/10.1145/228503.228517, Last visited [6 November 2010].

HP, 2010a, *HP Network Node Manager (NNM) Advanced Edition software*, URL https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-15-119^1155_4000_100__, Last visited [24 February 2010].

HP, 2010b, *Looking for HP OpenView?*, URL https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-10^36657_4000_100, Last visited [24 February 2010].

Hsin, C. & Liu, M., 2006, *Self-monitoring of wireless sensor networks*, Computer Communications, 29(4):462 – 476, URL http://www.sciencedirect.com/science/article/B6TYP-4FGX42D-1/2/5b540f4d631b53aebed3fa76279a28a0, Last visited [6 November 2010].

Hui, J. & Thubert, P., 2009, *Compression Format for IPv6 Datagrams in 6LoWPAN Networks*, draft -ietf-6lowpan-hc-06, Internet Engineering Task Force, URL `http://tools.ietf.org/html/draft-ietf-6lowpan-hc-06`, Last visited [7 November 2010].

Hui, J.W., 2008, *An Extended Internet Architecture for Low-Power Wireless Networks - Design and Implementation*, Ph.D. thesis, EECS Department, University of California, Berkeley, URL `http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-116.html`, Last visited [6 November 2010].

IEEE, 2004, *IEEE 802.15 WPAN Task Group 2 (TG2)*, URL `http://www.ieee802.org/15/pub/TG2.html`, Last visited [10 August 2011].

IEEE, 2006, *IEEE Standard for Information technology – Telecommunication and information exchange between systems – Local and metropolitan area networks – Specific requirements. Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*, IEEE Computer Society, URL `http://www.ieee802.org/15/pub/TG4.html`.

IPSO Alliance, 2010, *IPSO Alliance: Enabling the Internet of Things*, URL `http://www.ipso-alliance.org`, Last visited [6 November 2010].

ITU-T, 2008, *Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER) - Recommendation X.690*, URL `http://www.itu.int/itu-t/recommendations/rec.aspx?rec=x.690`, Last visited [6 November 2010], Identical standard: ISO/IEC 8825-1:2008 (Common).

ITU-T, 2010, *Introduction to ASN.1*, URL `http://www.itu.int/ITU-T/asn1/introduction/index.htm`, Last visited [6 November 2010].

Jacquot, A., Chanet, J.P., Hou, K.M., Diao, X. & Li, J.J., 2009, *LiveNCM : A new wireless management tool*, in *AFRICON, 2009. AFRICON '09.*, 1–6, IEEE.

Jang, H., Jeong, K. & Choi, D., 2006, *Delivery and Storage Architecture for sensing information using SNMP*, International Journal of Computer Science and Network Security, 6(4):130–134, URL `http://paper.ijcsns.org/07_book/html/200604/200604108.html`, Last visited [6 November 2010].

Jennic, 2010, *Choosing a network protocol stack*, URL `http://www.jennic.com/products/protocol_stacks/`, Last visited [6 November 2010].

Jennic, 2011, *Jennic Wireless Microcontrollers*, URL `http://www.jennic.com/`, Last visited [12 October 2011].

Kim, K., Mukhtar, H., Joo, S., Yoo, S. & Park, S.D., 2009, *6LoWPAN Management Information Base*, draft -daniel-6lowpan-mib-01, Internet Engineering Task Force, URL http://tools.ietf.org/html/draft-daniel-6lowpan-mib-01, Last visited [7 November 2010].

Kinney, P., 2010, *IEEE 802.15 WPAN Task Group 4 (TG4)*, URL http://www.ieee802.org/15/pub/TG4.html, Last visited [10 August 2011].

Kinney, P., 2011, *IEEE 802.15 WPAN SCwng Wireless Next Generation Standing Committee*, URL http://www.ieee802.org/15/pub/SCwng.html, Last visited [10 August 2011].

Krawczyk, H., Bellare, M. & Canetti, R., 1997, *HMAC: Keyed-Hashing for Message Authentication*, RFC 2104, Internet Engineering Task Force, URL http://www.rfc-editor.org/rfc/rfc2104.txt, Last visited [7 November 2010].

Kurner, T., 2003, *IEEE 802.15 WPAN Terahertz Interest Group (IGthz)*, URL http://www.ieee802.org/15/pub/IGthz.html, Last visited [10 August 2011].

Kushalnagar, N., Montenegro, G. & Schumacher, C., 2007, *IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals*, RFC 4919, Internet Engineering Task Force, URL http://www.rfc-editor.org/rfc/rfc4919.txt, Last visited [7 November 2010].

Langendoen, K., Baggio, A. & Visser, O., 2006, *Murphy Loves Potatoes: Experiences from a Pilot Sensor Network Deployment in Precision Agriculture*, in *Parallel and Distributed Processing Symposium, International*, 155, Los Alamitos, CA, USA: IEEE Computer Society, URL http://doi.ieeecomputersociety.org/10.1109/IPDPS.2006.1639412, Last visited [6 November 2010].

Lee, M., 2011, *IEEE 802.15 WPAN Task Group 5 (TG5) Mesh Networking*, URL http://www.ieee802.org/15/pub/TG5.html, Last visited [10 August 2011].

Lee, W.L., Datta, A. & Cardell-Oliver, R., 2007, *Network Management in Wireless Sensor Networks*, in M.K. Denko & L.T. Yang, editors, *Handbook of Mobile Ad Hoc and Pervasive Communication*, School of Computer Science & Software Engineering, The University of Western Australia, American Scientific Publishers, USA, URL http://www.csse.uwa.edu.au/~winnie/Network_Management_in_WSNs_.pdf, Last visited [6 November 2010].

LiteOS, 2010, *LiteOS Home*, URL http://www.liteos.net/, Last visited [6 November 2010].

Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R. & Anderson, J., 2002, *Wireless sensor networks for habitat monitoring*, in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, WSNA '02, 88–97, New York, NY, USA: ACM, URL http://doi.acm.org/10.1145/570738.570751, Last visited [6 November 2010].

McCloghrie, K., 1996, *An Administrative Infrastructure for SNMPv2*, RFC 1909, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc1909.txt`, Last visited [7 November 2010].

McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., McCloghrie, K., Rose, M. & Waldbusser, S., 1999a, *Conformance Statements for SMIv2*, RFC 2580, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc2580.txt`, Last visited [7 November 2010].

McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., McCloghrie, K., Rose, M. & Waldbusser, S., 1999b, *Structure of Management Information Version 2 (SMIv2)*, RFC 2578, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc2578.txt`, Last visited [7 November 2010].

McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., McCloghrie, K., Rose, M. & Waldbusser, S., 1999c, *Textual Conventions for SMIv2*, RFC 2579, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc2579.txt`, Last visited [7 November 2010].

McCloghrie, K. & Rose, M., 1988, *Management Information Base for Network Management of TCP/IP-based internets*, RFC 1066, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc1066.txt`, Last visited [7 November 2010].

McCloghrie, K. & Rose, M., 1990, *Management Information Base for Network Management of TCP/IP-based internets*, RFC 1156, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc1156.txt`, Last visited [7 November 2010].

McCloghrie, K. & Rose, M., 1991, *Management Information Base for Network Management of TCP/IP-based internets: MIB-II*, RFC 1213, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc1213.txt`, Last visited [7 November 2010].

McCulloch, J., McCarthy, P., Guru, S.M., Peng, W., Hugo, D. & Terhorst, A., 2008, *Wireless sensor network deployment for water use efficiency in irrigation*, in *Proceedings of the workshop on Real-world wireless sensor networks*, REALWSN '08, 46–50, New York, NY, USA: ACM, URL `http://doi.acm.org/10.1145/1435473.1435487`, Last visited [6 November 2010].

MEMSIC, 2011, *MEMSIC*, URL `http://www.memsic.com/`, Last visited [12 October 2011].

MikroTik, 2010, *MikroTik Routers and Wireless: The Dude*, URL `http://www.mikrotik.com/thedude.php`, Last visited [6 November 2010].

Montenegro, G., Kushalnagar, N., Hui, J. & Culler, D., 2007, *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*, RFC 4944, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc4944.txt`, Last visited [7 November 2010].

Mukhtar, H., Joo, S., Schoenwaelder, J. & Kim, K., 2009, *SNMP optimizations for 6LoWPAN*, draft -hamid-6lowpan-snmp-optimizations-02, Internet Engineering Task Force, URL `http://tools.ietf.org/html/draft-hamid-6lowpan-snmp-optimizations-02`, Last visited [7 November 2010].

Nagios, 2010, *The Industry Standard in IT Infrastructure Monitoring*, URL `http://www.nagios.org/`, Last visited [6 November 2010].

Nano-RK, 2010, *Nano-RK: A Wireless Sensor Networking Real-Time Operating System*, URL `http://www.nanork.org/`, Last visited [6 November 2010].

NIST, 1995, *Secure Hash standard*, URL `http://www.itl.nist.gov/fipspubs/fip180-1.htm`, Last visited [6 November 2010], Federal Information Processing Standards Publication 180-1.

Oetiker, T., 2009, *RRDtool - About RRDtool*, URL `http://oss.oetiker.ch/rrdtool/`, Last visited [6 November 2010].

Oetiker, T., 2011, *RRDTool - rrdtool*, URL `http://oss.oetiker.ch/rrdtool/doc/rrdtool.en.html`, Last visited [9 January 2011].

OpenNMS, 2010a, *FAQ-About*, URL `http://www.opennms.org/wiki/FAQ-About`, Last visited [6 November 2010].

OpenNMS, 2010b, *The OpenNMS Project*, URL `http://www.opennms.org`, Last visited [6 November 2010].

Padlipsky, M., 1982, *A perspective on the ARPANET reference model*, RFC 871, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc871.txt`, Last visited [7 November 2010].

Postel, J., 1980, *User Datagram Protocol*, RFC 768, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc768.txt`, Last visited [7 November 2010].

Postel, J., 1981, *Transmission control protocol darpa internet program protocol specification*, RFC 793, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc793.txt`, Last visited [7 November 2010].

Presuhn, R., Case, J., McCloghrie, K., Rose, M. & Waldbusser, S., 2002a, *Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)*, RFC 3418, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc3418.txt`, Last visited [7 November 2010].

Presuhn, R., Case, J., McCloghrie, K., Rose, M. & Waldbusser, S., 2002b, *Transport Mappings for the Simple Network Management Protocol (SNMP)*, RFC 3417, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc3417.txt`, Last visited [7 November 2010].

Presuhn, R., Case, J., McCloghrie, K., Rose, M. & Waldbusser, S., 2002c, *Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)*, RFC 3416, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc3416.txt`, Last visited [7 November 2010].

Rivest, R., 1992, *The MD5 Message-Digest Algorithm*, RFC 1321, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc1321.txt`, Last visited [7 November 2010].

Rose, M. & McCloghrie, K., 1988, *Structure and Identification of Management Information for TCP/IP-based internets*, RFC 1065, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc1065.txt`, Last visited [7 November 2010].

Rose, M. & McCloghrie, K., 1990, *Structure and Identification of Management Information for TCP/IP-based Internets*, RFC 1155, Internet Engineering Task Force, URL `http://www.rfc-editor.org/rfc/rfc1155.txt`, Last visited [7 November 2010].

Sensinode, 2010, *Products - Sensinode*, URL `http://www.sensinode.com/EN/products.html`, Last visited [6 November 2010].

Sensinode Ltd, 2011, *Sensinode Home - Sensinode Ltd*, URL `http://www.sensinode.com/`, Last visited [12 October 2011].

Sentilla Corporation, 2011, *Data Center Analytics Software - Sentilla*, URL `http://www.sentilla.com/`, Last visited [12 October 2011].

Shay, W.A., 2004, *Understanding data communications and networks - Third Edition*, Bill Stenquist - Thomson Brooks/Cole.

SNMPv3WG, 2002, *SNMP Version 3 (SNMPv3)*, URL `http://www.ietf.org/wg/concluded/snmpv3.html`, Last visited [6 November 2010], SNMP Version 3 Working Group.

Stefanov, I., 2008, *FS20 / SNMP Gateway on TinyOS*, Guided research final paper, Jacobs University Bremen, URL `http://www.eecs.jacobs-university.de/archive/bsc-2008/stefanovIvan.pdf`, Last visited [6 November 2010].

Szewczyk, R., Mainwaring, A., Polastre, J., Anderson, J. & Culler, D., 2004, *An analysis of a large scale habitat monitoring application*, in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, 214–226, New York, NY, USA: ACM, URL http://doi.acm.org/10.1145/1031495.1031521, Last visited [6 November 2010].

Thaler, D., 2002, *FW: ipv6IfStateChange traps in RFC2465*, Mailing list, URL http://www.ops.ietf.org/lists/v6ops/v6ops.2002/msg00269.html, Last visited [10 August 2011].

TinyOS, 2010, *TinyOS Home Page*, URL http://www.tinyos.net/, Last visited [6 November 2010].

Tolle, G. & Culler, D., 2005, *Design of an Application-Cooperative Management System for Wireless Sensor Networks*, in *Proceeedings of the Second European Workshop on Wireless Sensor Networks*, 121 – 132.

Tolle, G., Polastre, J., Szewczyk, R., Culler, D., Turner, N., Tu, K., Burgess, S., Dawson, T., Buonadonna, P., Gay, D. & Hong, W., 2005, *A macroscope in the redwoods*, in *Proceedings of the 3rd international conference on Embedded networked sensor systems*, SenSys '05, 51–63, New York, NY, USA: ACM, URL http://doi.acm.org/10.1145/1098918.1098925, Last visited [6 November 2010].

Vancea, C.M. & Dobrota, V., 2007, *Enabling SNMP for IEEE 802.15.4: A Practical Architecture*, in *6th RoEduNet International Conference "Networking in Education and Research"*, 49–53, URL http://www.csfnau.kiev.ua/kipz/ua/ROMANIA/papers/049%20-%20053%20-%20Vancea.pdf, Last visited [6 November 2010].

Waters, G., 1996, *User-based Security Model for SNMPv2*, RFC 1910, Internet Engineering Task Force, URL http://www.rfc-editor.org/rfc/rfc1910.txt, Last visited [7 November 2010].

WEBS, 2004, *nesC: A Programming Language for Deeply Networked Systems*, URL http://nescc.sourceforge.net/, Last visited [6 November 2010].

Werner-Allen, G., Johnson, J., Ruiz, M., Lees, J. & Welsh, M., 2005, *Monitoring Volcanic Eruptions with a Wireless Sensor Network*, in *Proceedings of the Second European Workshop on Wireless Sensor Networks*, EWSN05, 108 – 120.

Wijnen, B., Presuhn, R. & McCloghrie, K., 2002, *View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)*, RFC 3415, Internet Engineering Task Force, URL http://www.rfc-editor.org/rfc/rfc3415.txt, Last visited [7 November 2010].

Wijnen, B., 1996, *SNMPv2* compared to SNMPv2u*, URL http://www.simple-times.org/pub/simple-times/usec/v2compare.html, Last visited [18 October 2011].

Won, E.T., 2011, *IEEE 802.15 WPAN Task Group 7 (TG7) Visible Light Communication*, URL http://www.ieee802.org/15/pub/TG7.html, Last visited [12 October 2011].

Yu, M., Mokhtar, H. & Merabti, M., 2006, *A Survey of Network Management Architecture in Wireless Sensor Network*, in *Proceedings of the Sixth Annual PostGraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting*, URL `http://www.cms.livjm.ac.uk/pgnet2006/Programme/Papers/2006-093.pdf`, Last visited [6 November 2010].

Zenoss Community, 2010, *Open Source Network Monitoring and Systems Management*, URL `http://community.zenoss.org`, Last visited [6 November 2010].

Zigbee Foundation, 2010, *Zigbee Foundation Website*, URL `http://www.zigbee.org`, Last visited [6 November 2010].

Appendix A

Testbed

An indoor WSN testbed was constructed as part of this research to aid in testing the SNMPv1 software agent implementation. The main aim of the testbed was to provide a means to power the nodes and also to provide the ability to remotely reprogram the nodes used on the testbed. The programming of the nodes should be made as easy as possible.

A separate PC from the one running the *blip* routing driver and network monitoring software was used to control the testbed. This was done to facilitate the use of the testbed by other users after the completion of this research.

A.1 Hardware

The testbed uses Sentilla (previously Moteiv) Tmote Connect gateway appliances to connect the WSN nodes used to a wired ethernet, Figure A.1. The testbed wired ethernet is separate from the institutional LAN to eliminate any interference of one network on the other. The testbed PC has two network interface cards, one connected to the testbed LAN and the other to the institutional LAN. In this way the PC can be accessed remotely to reprogram the testbed while also keeping the two LANs separate.

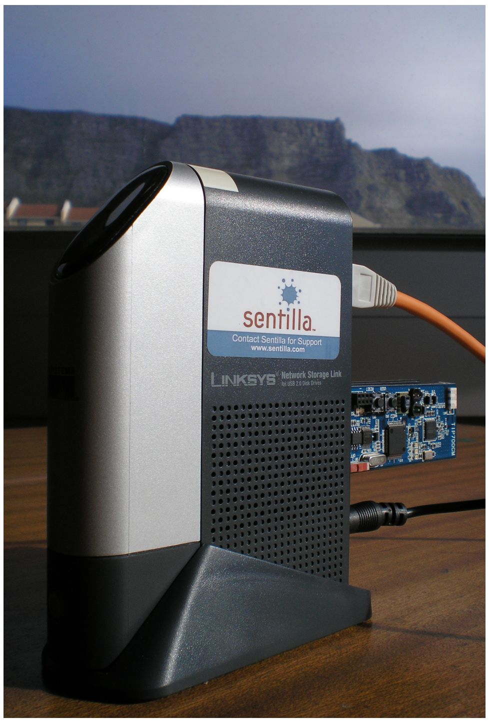


Figure A.1: Sentilla Tmote Connect with one WSN node

Each Connect has one TelosB platform WSN node attached to it, this can be adjusted as needed. At present some of the Connects house two nodes to compensate for two Connects that have failed, in order to keep the total amount of nodes the same.

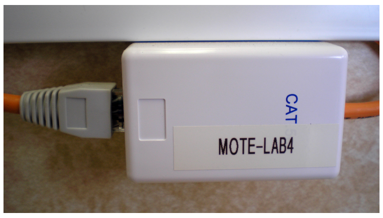


Figure A.2: Testbed network cable adapter box

Separate cabling was done for the testbed LAN and was placed in the trunking used by the institutional LAN and connector boxes used to terminate the cables, Figure A.2. An unmanaged network switch was used to connect the various network elements. A list of the current installed nodes and Connects can be found in the script that is used to reprogram the whole testbed. The location of each testbed LAN cable adapter box can be seen on Figure A.3, this corresponds to the IP address of the Connect and the ID of the node attached to it.

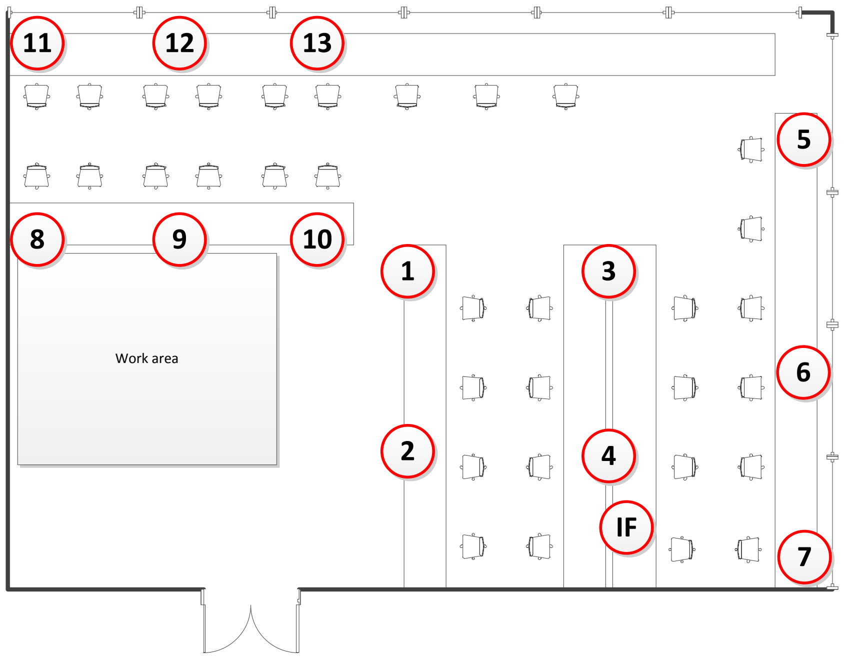


Figure A.3: Floor plan of testbed and node location

A.2 Software

Initially it was attempted to use the Motelab testbed software developed by Harvard. But this was abandoned after a considerable amount of time was spent to install and set up the software with only limited success. The solution developed is not as full featured as the Motelab software but it includes the needed features as mentioned above.

The core of the testbed software is the `netbsl` script used to program nodes connected to the Tmote Connect appliances. A script, `runNetbsl`, was created that uses the `netbsl` script to program a specific node on the testbed. A second script, `progLab`, was created that uses the `runNetbsl` script to reprogram the whole testbed. The `runNetbsl` script takes as arguments the host IP address of the Tmote Connect the node is connected to, the port the node is connected to and the TOS_NODE_ID to be assigned to the node.

```
$ ./runNetbsl 10.0.0.13 10001 13
```

The output from the script of a successful programming can be seen below:

```
------------>START_ID-13@10.0.0.13:10001
----->Mon Nov 8 16:51:21 SAST 2010
```

```
Using mote  on port /dev/ttyUSB0.
Mass erase.
Invoking BSL.
BSL version 1.61, MCU device id f16c.
Changing to 38400 baud.
Program image /tmp/_dev_ttyUSB0, 45294 bytes.
Invoking BSL.
BSL version 1.61, MCU device id f16c.
Changing to 38400 baud.
Program.
Programmed 45326 bytes.
Reset
------------>END_ID-13@10.0.0.13:10001
```

The `progLab` script takes a dummy argument to start programming the testbed. The nodes to be programmed is specified within the `progLab` script.

```
./progLab now
```

The `progLab` script is not very intelligent and will not attempt to reprogram a node that has not been successfully programmed. The output from the script should therefore be examined to verify that all nodes programmed successfully.

After successful compilation the `build` directory of the application to be programmed should be copied into the same directory as both scripts. The scripts can then be invoked to program the nodes.

The Tmote Connects also respond to commands directly. An example is retrieving status information on the node connected to a Connect. More commands can be found in the datasheet for the Sentilla Tmote Connect. It is possible to reset a node or to reboot the Connect if needed.

```
echo status | nc 10.0.0.1 10001
```

The `netbsl` and `netbsl.extra` scripts should be copied to the following directory of the TinyOS install: `$TOSROOT/support/make/msp/`.

Appendix B

Set up and configuration of software

This appendix chapter contains more details on the set up, configuration and use of the software mentioned in the main body of work. The configuration files for Nagios will be examined in more detail. The addition of devices and creation of new graphs in Cacti will be covered. The setup used for PHP graphing will also be covered in more detail.

B.1 Nagios

Unlike Cacti, all configuration for Nagios is done in the configuration files found under `\etc\nagios3\`. The whole `nagios3` directory with the configuration files used can be found on the accompanying DVD.

Hosts can be defined by placing a host definition in the `hosts.cgf` file. An example of a host definition can be seen below:

```
define host{
  host_name  mote1
  alias      TelosB node 1
  address    fec0::1
  parents    WSNrouter
  use        generic-host
  hostgroups all
}
```

When a host is added it should also be added to one or more host groups in the `hostgroups.cgf` file.

```
define hostgroup {
  hostgroup_name    snmp-voltage
    alias           Nodes to check for voltage
    members         mote1, mote2, mote3, mote4, mote5, mote6, mote7
}
```

The `services.cfg` file contains definitions on the services to be checked and what host group the service check is associated with. Two new commands used in the `services.cfg` file were defined in the `commands.cfg` file, these include the `snmp_node_voltage` and `snmp_node_uptime commands`. These commands retrieve the corresponding information using the SNMP. The levels at which warnings and critical warnings are given in Nagios are set in the `services.cfg` file, this can be seen as arguments given when the command is invoked. This can be seen below:

```
check_ping!5000.0,50%!7500.0,70%!1!7000!-6
snmp_node_voltage!public!2500!3500
snmp_node_uptime!public
```

The first argument of the `check_ping` command is the values for RTA and percentage packet loss at which a warning is given. The second is the values at which a critical warning is given. The third argument is the number of packets to send while the fourth specifies the timeout between packets (if more than one is sent). The last argument indicates that IPv6 is to be used. The levels for warnings from this service check should be adjusted to values more suitable for WSNs, which is typically longer timeout values and longer RTAs and slightly higher packet loss should be tolerated.

For both SNMP commands the first argument is the community string of the community that the nodes belongs to. For the `snmp_node_voltage` command it is possible to specify warning levels, with the first indicating a warning and the second indicating a critical warning.

Once configuration is complete a ‘pre-flight’ check can be done on the configuration data to ensure that it is correct:

```
$ sudo nagios3 -v /etc/nagios3/nagios.cfg
...
Total Warnings: 0
Total Errors:   0

Things look okay - No serious problems were detected during the pre-flight check
```

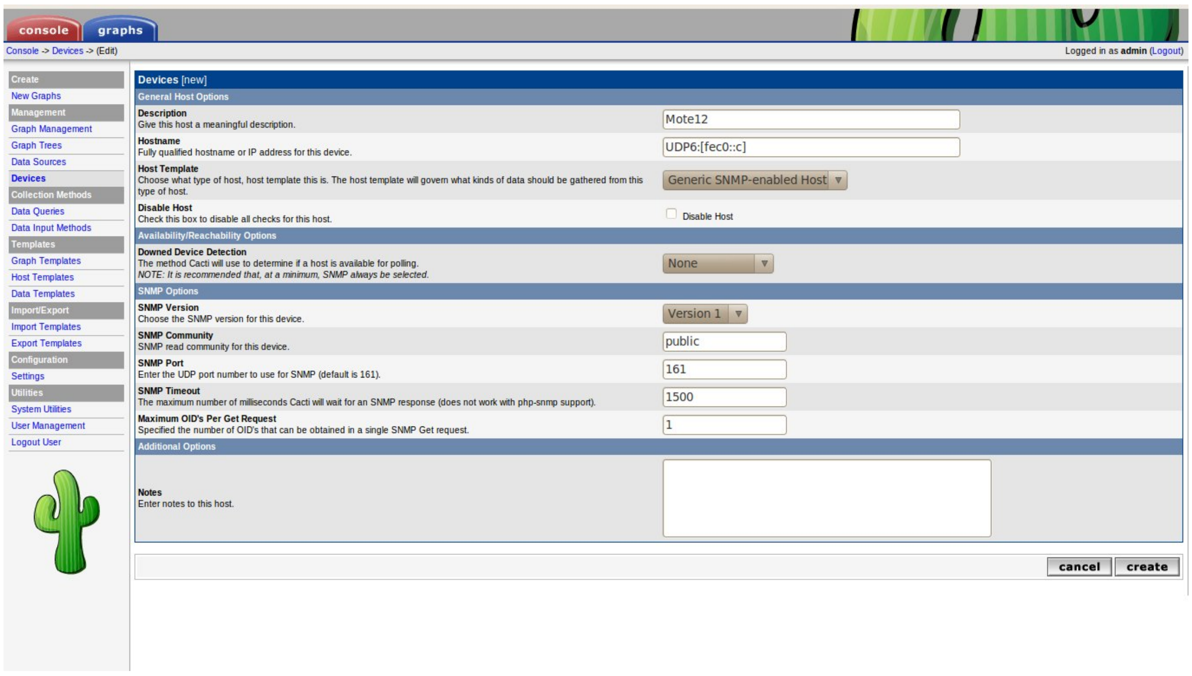
If all is well Nagios can be restarted for the changes to take effect:

```
$ sudo /etc/init.d/nagios3 restart
```

B.2 Cacti

The setup and configuration of Cacti is done completely through the web interface, under the *console* tab. Before graphs can be created devices need to be added to Cacti. This is done under the *Devices* page under the *Management* heading on the left hand panel. To add a device the user needs to select *Add* from the top right corner of the *Devices* page. On this page the details of the node can be entered.

A *Description* of the node should be entered in the first box with the nodes IPv6 address in the second. The SNMP options should be changed to accommodate the WSN setting the software is used in. The *SNMP Timeout* is increased to 1500 and the *Maximum OID's Per Request* set to one. The completed page can be seen in Figure B.1. The create button can then be selected to continue. On the next page under *Associated Data Queries* the red cross should be clicked to remove the *SNMP - Interface Statistics* item. The changes can then be saved to by selecting the *save* button.



Devices [new]

General Host Options

Description Give this host a meaningful description.	Mote12
Hostname Fully qualified hostname or IP address for this device.	UDP6:[fec0::c]
Host Template Choose what type of host, host template this is. The host template will govern what kinds of data should be gathered from this type of host.	Generic SNMP-enabled Host
Disable Host Check this box to disable all checks for this host.	☐ Disable Host

Availability/Reachability Options

Downed Device Detection The method Cacti will use to determine if a host is available for polling. *NOTE: It is recommended that, at a minimum, SNMP always be selected.*	None

SNMP Options

SNMP Version Choose the SNMP version for this device.	Version 1
SNMP Community SNMP read community for this device.	public
SNMP Port Enter the UDP port number to use for SNMP (default is 161).	161
SNMP Timeout The maximum number of milliseconds Cacti will wait for an SNMP response (does not work with php-snmp support).	1500
Maximum OID's Per Get Request Specified the number of OID's that can be obtained in a single SNMP Get request.	1

Additional Options

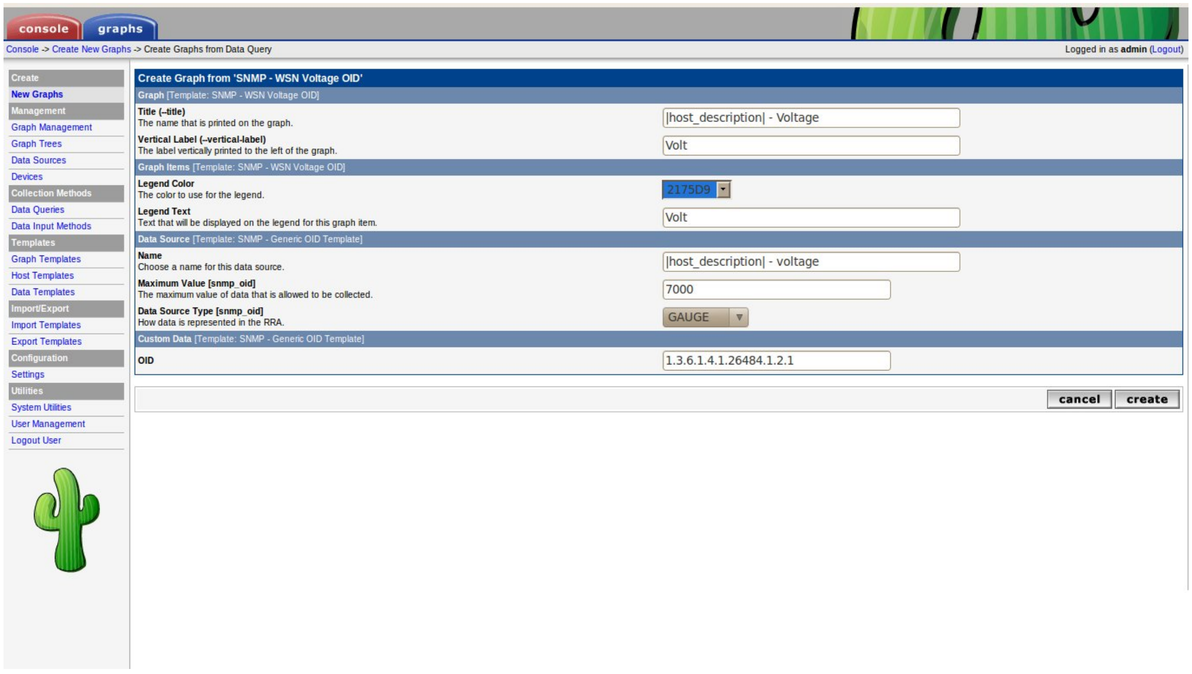
Notes Enter notes to this host.	

Figure B.1: Adding a device

Graphs for this device can now be created. The graph templates created and used in the research can be found on the accompanying DVD and imported to Cacti for use. It also possible to edit and change templates to better suite the intended application. These graph templates will we used to create graphs for the node that was created. This done by selecting *New Graphs* from the left hand panel under the heading *Create*.

Select the device for which a graph should be created from the drop down menu for *Host:* and select the *create* button. After the device is selected a graph template can be selected from the drop down menu under *Graph Templates*, *Graph Template Name* and *Create:*. For demonstration the *SNMP - WSN Voltage OID* graph template will be used. After the graph template is selected the *create* button should be selected.

Figure B.2 shows the next completed page. On this page the title of the graph and name of data source should be entered and also details on the SNMP query. The *Maximum Value [snmp_oid]* field should contain a value that is higher than the maximum value that is expected to be returned by the node, otherwise the values will be ignored and not graphed. The OID for the object that contains the voltage reading should be entered into the *OID* field. Select the *create* button.



Create Graph from 'SNMP - WSN Voltage OID'

Graph [Template: SNMP - WSN Voltage OID]

Title (--title) The name that is printed on the graph.	|host_description| - Voltage
Vertical Label (--vertical-label) The label vertically printed to the left of the graph.	Volt

Graph Items [Template: SNMP - WSN Voltage OID]

Legend Color The color to use for the legend.	2175D9
Legend Text Text that will be displayed on the legend for this graph item.	Volt

Data Source [Template: SNMP - Generic OID Template]

Name Choose a name for this data source.	|host_description| - voltage
Maximum Value [snmp_oid] The maximum value of data that is allowed to be collected.	7000
Data Source Type [snmp_oid] How data is represented in the RRA.	GAUGE

Custom Data [Template: SNMP - Generic OID Template]

OID	1.3.6.1.4.1.26484.1.2.1

Figure B.2: Creating the graph

The created graph should now be placed on one of the trees so that it can be viewed. This is done under *Graph Management* under the *Management* heading on the left hand panel. The graph to placed on the graph tree should be selected and at the bottom of the page from the *Choose an action* drop down menu the *Place on a Tree* option corresponding to the wanted tree should be selected and the *go* button pressed. If the tree has multiple branches the *Destination Branch* can be selected and the *yes* button selected. The graph can then be viewed from the *graphs* tab and by selecting the appropriate tree and branch from the left hand panel.

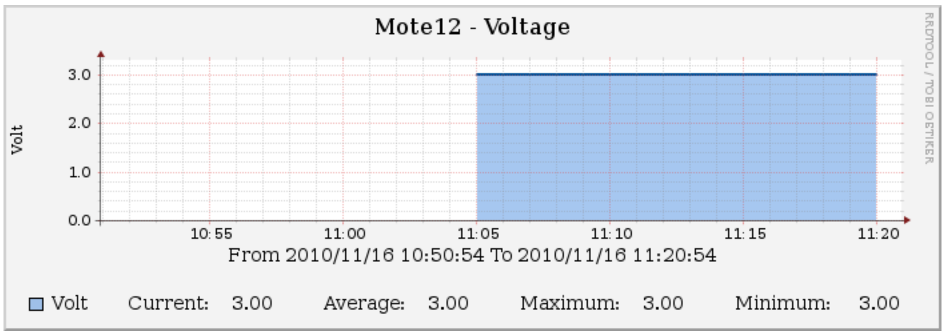


Figure B.3: The new graph created from the device added

Keep in mind that it can take up to ten minutes for the graph to be populated with data, depending on the settings used in Cacti. The graph created from the device that was added can be seen in Figure B.3.

B.3 PHP graphing

The graphing was done using PHP and the JpGraph graphing library. The installation under Ubuntu is straightforward and the Synaptic Package Manager can be used to install the needed packages. The extra packages needed are: `python-mysqldb`, `libphp-jpgraph` and `libphp-jpgraph-examples`. After the packages are installed it is needed to restart Apache:

```
$ sudo /etc/init.d/apache2 restart
```

Before looking at the graphing scripts it should be noted how the back-end to the graphing operates. The data is stored in a MySQL database by a Python script. The Python script is a slightly modified version of the script included in the UDPEcho demo application of blip and can be found in `$TOSROOT/apps/UDPEcho/util`. The actual script used is included with the SNMPv1 Agent code under `apps/windStation/scripts`. A user and database with the correct permissions should be added to the MySQL database. The `MySQLListenerV2.py` script can be used to log incoming messages to the database or the `senseListener.py` can be used to only display the received data to the screen. The scripts should be run using the following syntax:

```
$ python scriptName.py
```

The following line should be added to the `tinyos.sh` file under `$TOSROOT` if it is not already present:

```
export PYTHONPATH="$PYTHONPATH:$TOSROOT/support/sdk/python"
```

The PHP scripts created are included on the accompanying DVD, with the whole `/var/www` folder included on the disc. Care should be taken that the files and directories have the appropriate permissions. The `index.htm` file contains links to the Nagios, Cacti and PHP graph pages. The PHP graph scripts can be found under `blip_deployment_graphs`. This directory contains an `index.php` and `include.php` script and also a directory with the graph scripts for the different sensor values to be graphed.

The `include.php` script contains information on the database that is used by the graphing scripts including the tablename, username, password, host address and database name. The `index.php` script has links to graphs for different time periods, 24 hours and three days. The ID of the node to be graphed is also set within this script. The script makes use of the `landing.php` script in the `genericGraphs` directory to create a single page with all the graphs for the node. The various graph scripts and `landing.php` accept the same mandatory arguments. This means that the `landing.php` script can be used to show all graphs or the script for a single sensor can be used.

```
http://<host>/blip_deployment_graphs/genericGraphs/landing.php?nodeID=13&period=1
http://<host>/blip_deployment_graphs/genericGraphs/voltage.php?nodeID=13&period=1
```

The `index.php` script passes the appropriate arguments to the graphing scripts, these include `nodeID` to indicate the ID of the node to be graphed and `period` for the time period to be graphed.

Appendix C

Code

All source code, scripts, Nagios configuration files and Cacti templates can be found on the accompanying DVD.

Document prepared using LaTeX