LIGHTWEIGHT NETWORK MANAGEMENT DESIGN FOR WIRELESS SENSOR NETWORKS

By

FENGHUA YUAN

A thesis submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE

WASHINGTON STATE UNIVERSITY
School of Engineering and Computer Science

DECEMBER 2007

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of FENGHUA YUAN find it satisfactory and recommend that it be accepted.

Chair

ACKNOWLEDGEMENT

I would like to express my gratitude to my advisor, Professor WenZhan Song, for his guidance, inspiration and supervision of this thesis. Thanks for his useful comments to improve my thesis.

I would also express sincere appreciation to all the members on our team, who gave me so many good suggestions when I did the implementation part for this thesis.

Many thanks to all my friends at WSU. Without their generous help, continuous encouragement and moral support, this work could not have been completed.

FENGHUA YUAN

LIGHTWEIGHT NETWORK MANAGEMENT DESIGN

FOR WIRELESS SENSOR NETWORKS

Abstract

by Fenghua Yuan, M.S.
Washington State University
December 2007

Chair: WenZhan Song

Wireless sensor networks (WSNs) play an increasingly important role in supporting a wide range of applications for our daily life, such as disaster relief and environment control. They also pose certain design and implementation challenges. One of the biggest challenges is the design of an efficient network management system to provide management services to support WSNs for various sensor applications. The unique features of WSNs, such as limited capacities of nodes, unreliable communication media, and high diversity of applications, make the design of Sensor Network Management System (SNMS) very different from traditional networks.

This thesis first proposes the general principles of how to design a management system for WSNs. Then a lightweight SNMS is designed. It is called the “lightweight” SNMS because of two reasons: First, the SNMS occupies a minimal amount of RAM and generates less network traffic, which limit the management overhead and meet the resource constrains of sensor nodes. Second, it requires a minimal effort from the application developer to add the SNMS into the application. The generic framework of the SNMS allows the SNMS to be decoupled with the application, and be transparent for the application. Since the SNMS is non-application-oriented, it decreases the development overload for different applications. In this thesis, the RPC (Remote Procedure Call) based SNMS architecture is presented, and the design of main functions, including performance management, fault management, and configuration management, are described in detail.

Following the lightweight SNMS design proposed in this thesis, a generic SNMS is implemented. It works in the client/server paradigm. The SNMS server is developed in TinyOS/nesC language, and can be easily wired, through its generic interface, to the other modules of the application running on the sensor node. The SNMS client is a java tool suite running on a PC. It provides remote monitor and control and remote reprogramming for WSNs through a friendly GUI. The testing of this lightweight SNMS is done on a 17-mote MICAz/MIB600 TestBed to show the validity and efficiency of this SNMS design.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

1.1 Motivation

1.1.1 Wireless Sensor Networks

Recall the computer revolutions, the one of the 1980s was the advent of the personal computer, which gave rise to cheap computers, and made them become ubiquitous in homes, schools and offices. The computer revolution of the late 1990s, now continuing to the 21st century, involves computer networks, whose existence is enriching our society in countless different ways, such as providing a good communication medium, sharing of available resources, improved reliability of service, and cost-effectiveness [28].

A computer network consists of two major components: distributed applications and networking infrastructure. The networking infrastructure provides support for data transfer among the interconnected computers where the applications reside. Wireless radio is one of the physical media used in the present-day communication infrastructure, and the networks using wireless radio waves are called wireless networks accordingly.

There are two categories of wireless networks based on whether they depend on the support of any fixed infrastructure [40]. For example, the current cellular wireless networks are classified as the infrastructure dependent networks; ad hoc wireless networks are defined as the category of wireless networks that utilize multi-hop radio relaying, and are capable of operating without the support of any fixed infrastructure.

Wireless Sensor networks (WSNs) [3] are a special category of ad hoc wireless networks (shown in Figure 1.1). They are highly distributed networks of small, lightweight wireless nodes, deployed in large numbers to monitor the environment or system by the measurement of physical parameters such as temperature, pressure, or relative humidity.

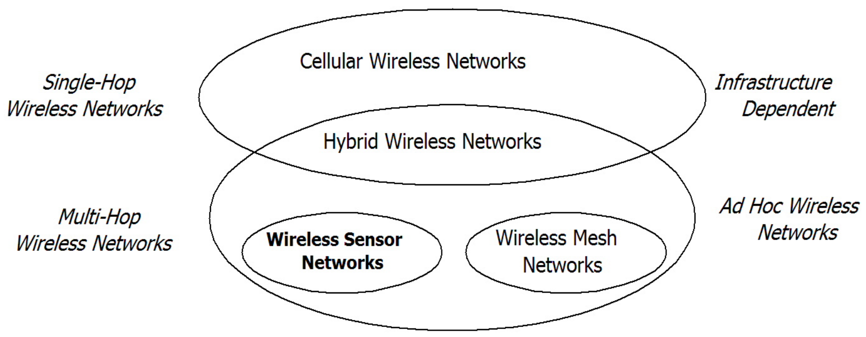


Figure 1.1: Wireless Networks

Wireless Sensor networks combine micro electromechanical systems (MEMS) technology, new sensor materials, low-power signal processing, computation, and low-cost wireless networking in a compact system. The features that make sensor networks a distinct category of ad hoc wireless networks are the following [3]:

- **Mobility of nodes** Mobility of nodes is not a mandatory requirement in sensor networks.
- **Size of the network** The number of nodes in the sensor network can be much larger than that in a typical ad hoc wireless network.
- **Density of deployment** The density of nodes in the sensor network varies with the domain of application.
- **Power constraints** The power constraints in sensor networks are much more stringent than those in ad hoc wireless networks. This is mainly because the sensor nodes are expected to operate in harsh environmental or geographical conditions, with minimum or no human supervision and maintenance.
- **Data/information fusion** The limited bandwidth and power constraints demand aggregation of bits and information at the intermediate relay nodes that are responsible for relaying. Data fusion refers to the aggregation of multiple packets into one before relaying it. They aim at reducing the bandwidth consumption.

- **Traffic distribution** The communication traffic pattern varies with the domain of application in sensor networks.

Wireless Sensor networks are used in a variety of applications which require constant monitoring and detection of specific events [34],[24]. The military applications of sensor nodes include battlefield surveillance and monitoring, guidance systems of intelligent missiles, and detection of attack by weapons of mass destruction, such as chemical, biological, or nuclear. Sensors are also used in environmental applications such as forest fire and flood detection, and habitat exploration of animals. Sensors can be useful in patient diagnosis and monitoring too. The applications of sensor networks are endless, limited only by the human imagination. With the increasing development of electronic components and advances in communication technology, wireless sensor networks have an enormous economical potential in future.

Wireless Sensor Networks have the potential to benefit our society in numerous ways [14]. At the same time, they also pose many scientific challenges due to the following reasons [40]:

- Sensor networks are infrastructure-less. Therefore, all routing and maintenance algorithms need to be distributed. An important bottleneck in the operation of sensor nodes is the available energy.

- Hardware design for sensor nodes should also consider energy efficiency as a primary requirement. Sensor nodes should be able to synchronize with each other in a completely distributed manner.

- A sensor network should also be capable of adapting to changing connectivity due to the failure of nodes, or new nodes deployed.

- Real-time communication over sensor networks must be supported through provision of guarantees on maximum delay, minimum bandwidth, or other QoS parameters.

- Sensor nodes are randomly deployed and hence do not fit into any regular topology. Once deployed, they usually do not require any human intervention. Hence, the setup and maintenance of the network should be entirely autonomous.

To meet these requirements, innovative mechanisms for wireless sensor networks have to be found, as well as new architectures and protocol concepts. Some of the mechanisms that will form typical parts of WSNs are [23]:

- **Multihop wireless communication** While wireless communication will be a core technique, there is a limitation of the direct communication between a sender and a receiver. The use of intermediate nodes as relays can reduce the total required power. Hence, for many forms of WSNs, so-called *multihop communication* will be a necessary ingredient.

- **Energy-efficient operation** It is a critical technique to support long lifetimes of WSNs. Options to look into include energy-efficient data transport between two nodes, or, more importantly, the energy-efficient determination of requested information.

- **Auto-configuration** A WSN will have to configure most of its operational parameters autonomously, independent of external configuration, which is the capability required in most applications. For example, self-location, the ability to tolerate failing nodes, or to integrate new nodes.

- **Collaboration and in-network processing** In some applications, several sensors have to collaborate to detect an event and only the joined data of many sensors provides enough information. Information is processed in the network itself to achieve this collaboration.

- **Data centric** Different from traditional address-centric (each device equipped with at least one network address) network, in a WSN, the nodes are typically deployed redundantly to protect against node failures or to compensate for the low quality of a single node's actual sensing equipment. What is important is the data, not which node has provided the data, thus it is called a data-centric network.

1.1.2 Necessity of Network Management in WSNs

During the past few years, a lot of research efforts have focused on technologies of networking these tiny sensor devices in wireless sensor networks, such as energy efficient media access control (MAC) protocol, topology control protocols and routing schemes. And usually, WSNs and their applications using these techniques have been developed without considering a management solution.

In general, WSNs are likely to operate under very dynamic and critical environments with applications such as environment monitoring, public safety, medical, transportation and military. Sensor nodes are usually difficult to access because of the geographical locations where they are deployed or the large scale of the network. Thus, network maintenance for reconfiguration, recovery from failure or technical problems become impractical [48].

In early days, this problem could be ignored as WSNs were suppose to operate cheaply and nodes were disposable. If the system breaks or underperforms, more nodes would be deployed to cover the failure. With the development of more and more powerful sensor nodes, this vision is not always the case. In addition, we would like to get the most out of the system we have at hand, which requires the WSN to cooperate with various applications rather than setting up a new network environment for each application. In the future, multiple applications may be required to be concurrently executed over a single network. Therefore, WSNs are indeed in need of some sort of management mechanism.

Management of WSNs is a new research area that only recently started to receive attention from the research community [30]. All the factors mentioned above make managing a WSN necessary. Another expectation for the management mechanism in a WSN involves a disaster scenario where the management software should be able to provide a real time network topology of the deployed nodes without requiring checking by a human, which is expensive and, most of time, is even impossible. At the same time, the WSN, running any application, should be able to reveal to a human manager whether a deployed network is functioning. WSN should also be able to record node failure events for post-mortem analysis, and this event record should also be accessible to a human manager in real time.

Based on the requirements for the management mechanism mentioned above, the Sensor Network Management is described as following: *Sensor Network Management is a system or process that seeks to manage and coordinate the sensor nodes in a dynamic and uncertain environment, to accomplish specific mission objectives and improve the performance of perception, by using the least amount of energy.*

1.1.3 Challenges for Network Management in WSNs

The research challenges for WSNs make the network management design for WSNs much more difficult than for traditional wired networks [35], [33].

- The management approaches must take the limited capabilities of the sensor nodes into account. The hardware limitations involving processor, memory, etc., with varying capabilities may play a significant role in determining the management capabilities. The weak processor (compared to those of a PC) limits the complexity of management functionality, and require it to be as simple as possible, but still reach the goal of monitoring and controlling the WSNs; The limited storage capacity will not allow the extra code for management module to occupy too much memory. So "lightweight" is one of the most important requirements when designing management system for WSNs. The challenge is how to balance the management protocol performance against its overhead.

- The transmission media related factors are important since transmission over the air in an unpredictable environment with possibly intentional or unintentional interference impact the link quality. The reliability of the transmission of management messages is another challenge in management design for WSNs. Meanwhile, constrained wireless bandwidth requires that a minimal amount of management messages be sent.

- The topology may also be an important factor in managing wireless sensor networks. Nodes may be deployed by dropping them from a plane, and are expected to dynamically self-configure depending on the environment to form the topology of the network. The topology may also be quite dynamic due to the failure or newly joined nodes, thus, automated reconfiguration is necessary, too.

- The WSNs are employed for a diverse set of applications ranging from military battlesite networks to disaster relief applications. These different applications have different management requirements, and the management system should be able to adapt easily to the characteristics of the situation.

1.2 Research Background

The research topic of this thesis comes from the Optimized Autonomous Space-In-situ Sensorweb (OASIS) project funded by NASA [1]. The goal of OASIS project is to develop a prototype dynamic and scalable hazard monitoring *Sensorweb*.

A *Sensorweb* is a coordinated observation infrastructure composed of a distributed collection of resources (e.g. sensors, platforms, models, communications infrastructure) that can collectively behave as a single, autonomous, task-able, dynamically adaptive and reconfigurable observing system that provides raw and processed data, along with associated meta-data, via a set of standards-based service-oriented interfaces.

Sensorweb related research spans multiple domains, including distributed systems, wireless sensor networks, remote sensing, artificial intelligence, sensor web services and etc. A multidisciplinary team involving computer scientists, space scientists, and earth scientists will work on this project. The final self-configuring/ self-healing remote Sensorweb built by this project will be applied to one of the most active volcanos - Mount St. Helens.

Among the environment monitoring applications, an erupting volcano provides a very challenging environment to examine and advance in-situ Sensorweb technology. The crater at Mount St. Helens is a dynamic 3-dimensional communication environment. After the sensor nodes are deployed in the crater area, various geophysical and geochemical sensors generate continuous high-fidelity data, whose priority depends on volcano status. There is a compelling need for real-time data, and sensors are destroyed occasionally by the eruption. Hence, an in-situ network must be self-configuring and self-healing, with a smart power and bandwidth management scheme, and autonomous in-network processing. The Figure 1.2 shows the in-situ network structure after the deployment of sensor nodes.

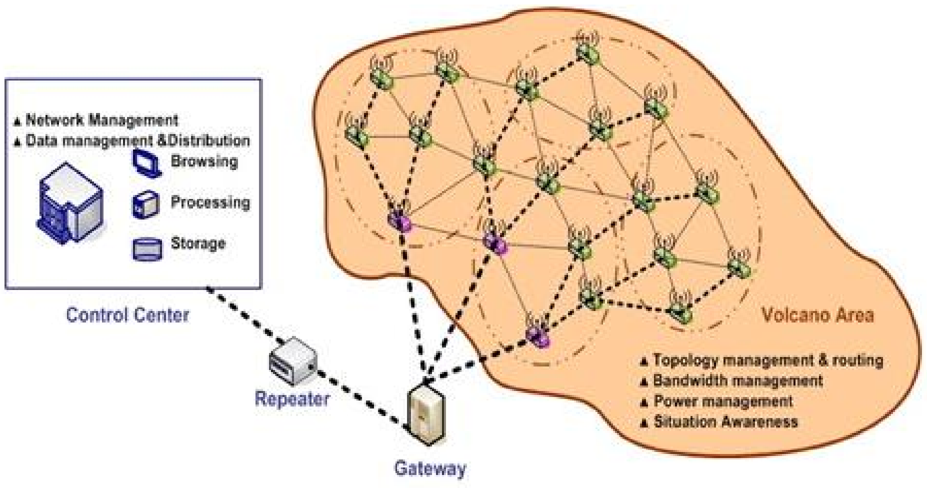


Figure 1.2: In-Situ Sensor-web Architecture

As mentioned in section 1.1.2, the harsh environment in the crater of volcano makes the management function more necessary and important for this project. The Sensor Network Management System should provide a way to monitor and control the WSN wirelessly from the control center, instead of the human control in place. The main requirements for the management of In-Situ wireless sensor network involve the following functional areas:

- Monitor the connectivity and status of network resources and provide indication of node failures, resource depletion, and other abnormality. Such information can provide early warning of system failure, and guidance for incremental deployment.

- Wirelessly reprogram the sensor nodes after the deployment when it is needed.
- Provide self-detect and self-healing mechanism for sensor nodes when some software or hardware faults happen, and also provide a mechanism for modules of application to report and log logic errors for the postmortem analysis.

1.3 Our Goal

As described in previous section, the objective of a WSN is to monitor and, eventually, control a remote environment. For WSN management, the objective is to define a set of functions that intend to promote productivity, as well as to integrate functions of configuration, operation, administration in an organized way, and maintenance of all elements and services of a sensor network [36]. Since network management for WSN is a relatively new research area, there are not many existing rules to follow or experiences to refer to.

One main goal of this thesis is to investigate the related problems when design such a Sensor Network Management System (SNMS), and seek to answer the following questions:

- What rules need to be followed specifically for designing a SNMS?
  Could those rules for designing a management system for traditional wired networks be applied directly to design a SNMS? What properties of WSNs need to consider when setting up these design rules?
- What network management functions need to define according to the requirements of sensor networks?
  Considering unique characteristics that WSNs have, such as wireless communication between two nodes, diverse densities for different applications, and dynamic topology, what

are the important functions that the SNMS for WSNs should have? How to design the SNMS for different applications running on the sensor nodes?

- How to design a lightweight SNMS?
  The "lightweight" has several meanings here: First, the SNMS should be designed to occupy a minimal amount of RAM and generate less network traffic in order to limit the management overhead and get maximized overall performance. Second, adding SNMS into an application of WSNs should require as less effort as possible from the developer of the application. The SNMS should be designed to be decoupled with the application, and be transparent for the developer.

In order to test the design rules which will be proposed, another goal of this thesis is to implement a generic SNMS module, which can be easily combined into the application running on a WSN to provide a management mechanism for the whole network.

Based on the generic SNMS module developed in this thesis, a real testing of our SNMS will be done on a in-door WSN TestBed to verify its correctness, completeness, and validation. It will show that the generic SNMS module can be easily customized to meet the requirements of management system for OASIS project. And the work done in this thesis will be used as the prototype of the management system for the OASIS project.

1.4 Thesis Outline

Wireless Sensor Networks (WSNs) are an emerging new research area in distributed computing. It plays an increasingly important role in supporting a wide range of applications for our daily life. It also poses certain design and implementation challenges. This thesis focuses on Sensor Network Management System (SNMS), one of the most challenging topics of WSN research. The necessity of SNMS for WSNs has been analyzed in Chapter 1, and the goal of design and implement a lightweight SNMS has also been proposed.

The following chapters will start from the background theories of developing a network management system, then explain how to achieve the goals listed in section 1.3 step by step.

Chapter 2 recalls the history of the network management development, and lists the network management standards defined by ISO as the guide of SNMS design. By comparing with the network management of traditional wired networks, the differences of SNMS design are analyzed. After the summary of related works on SNMS, the development principles of SNMS are proposed.

Chapter 3 follows the principles described in Chapter 2, and presents the idea of taking advantage of Remote Procedure Call (RPC) techniques to design a lightweight SNMS. The corresponding software architecture is provided, and the way to define functions under this lightweight framework is explained. Furthermore, the design of main functions of SNMS are provided in detail.

Chapter 4 goes one more step based on the work in Chapter 3, and implements a generic Sensor Network Management module. The developed SNMS consists of two parts: the SNMS server, located on the sensor node, and the SNMS client, running on a PC in the control center. The main techniques adopted to develop this SNMS, such as MoteIF communication interface and RPC mechanism are explained in detail.

Chapter 5 presents an example of how to use the SNMS designed in this thesis to manage a sensor network. The scenario of OASIS project (mentioned in Chapter 1) is taken as the application that needs to be managed. After adding the customized SNMS to it, the enhanced application is downloaded into the sensor nodes of a TestBed, and the functionality and validity of the SNMS module are tested and evaluated.

Chapter 6 concludes the thesis. It discusses the advantages and shortcomings of current design and implementation, and also notes several areas of future work.

CHAPTER TWO

SENSOR NETWORK MANAGEMENT

2.1 Network Management Standards

Recalling the development of management of traditional networks, there are several organizations that have developed services, protocols and architectures for network management. The three most important organizations are: The International Standards Organization (ISO), the International Telecommunication Union (ITU), and the Internet Engineering Task Force (IETF). Of these three ISO was the first who started, as part of its "Open Systems Interconnection"(OSI) program, the development of an architecture for network management [32].

The first proposal for such an architecture appeared during the early 1980's. The initial aim of this proposal by ISO was to define management standards for datacom networks, and the recommendations were self standing. During 1988 - 1992 study period, these recommendations have been rewritten to include the ideas of OSI management. Based on the discussion of the OSI management framework, IETF was requested to define an ad hoc management protocol, Simple Network Management Protocol (SNMP), which has become the de facto standard for network management nowadays.

2.1.1 Definition

According to the definition by ISO, *"Network management includes the deployment, integration, and coordination of the hardware, software, and human elements to monitor, test, poll, configure, analyze, evaluate, and control the network and element resources to meet the real-time, operational performance, and Quality of Service requirements at a reasonable cost."* [23]

The ISO has also created a network management model that is useful for placing all scenarios in a more structured framework. Five areas of network management are defined in that model [28]:

- **Performance management**

Quantify, measure, report, analyze, and control the performance of different network components. Protocol standards such as the *Simple Network Management Protocol* (SNMP) [RFC 3410] play a central role in Internet performance management.

- **Fault management**

  Log, detect, and respond to fault conditions in the network. Fault management is the immediate handling of transient network failures, while performance management is the longer term of performance. SNMP plays a central role in fault management too.

- **Configuration management**

  Allows a network manager to track which devices are on the managed network and the hardware and software configurations of these devices.

- **Accounting management**

  Allows specifying, logging and controlling user and device access to network resources. Usage quotas, usage-based charging, and the allocation of resource-access privileges all fall under accounting management.

- **Security management**

  Control access to network resources according to some well-defined policy. The main components of security management includes the key distribution centers and certification authorities, and the use of firewalls to monitor and control external access points to one's network.

First, the management should initialize the network system *(configuration management)*. If no errors occur, the network comes into service and the operational phase starts. During this phase, the management monitors the network system to check errors. In case of failures, the malfunctioning system will be identified, isolated and repaired. If the system can not be repaired, it will be replaced by a new system, which also must be initialized *(fault management)*. New systems may also be added to allow the connection of new users, to increase performance or to add new functionality. The addition of the new system usually implies reconfiguration. Monitoring the network is also useful to detect changes in the traffic flow. Once such changes are detected, network parameters may be modified to optimize the network's performance *(performance management)*.

2.1.2 Architecture

Based upon the information collection and communication strategy, there are three types of network management architectures: Centralized, Distributed, and Hierarchical[11], [48], [25].

In a centralized network management system, management decisions will be taken from the limited number of central locations. The management functionality that takes these decisions is called the manager. To manage the operation of the primary functions, agents should be added to the systems that perform primary functions. Such agents represent the management support functionality through which manager(s) initialize, monitor and modify the behavior of the primary functions. To allow managers to communicate with their agents, a management information protocol is necessary. Examples of such protocols are *Common Management Information Protocol* (CMIP) and *Simple Network Management Protocol* (SNMP).

With distributed management, there are no central systems from which management decisions are taken. Instead, functions that take such decisions will be added to the systems that already perform the primary functions. Such addition will usually be performed on a proportional scale. A distributed management system has multiple manager stations; each manages a subnetwork and communicates with other manager stations in a peer-to-peer manner. This approach has been adopted by the emphTelecommunication Management Network (TMN) and management model for emphAsynchronous Transfer Mode (ATM) networks [22].

Hierarchical network management systems use intermediate managers to distribute the manager tasks. Each intermediate manager has its domain; it collects and processes node information of its domain and passes the information to the upper level manager if necessary. It also distributes the messages from the upper level manager to nodes in its domain. There is no direct communication between intermediate managers.

A disadvantage of distributed management is that it will be difficult to change after the operational phase has started the functionality that makes the management decisions, because such changes require the modification of a large number of network systems, which will be expensive. It would be better to use the centralized management approach and concentrate the management functionality that makes the decisions within a single system. It is also easier to introduce Intelligent Networks when using centralized management.

A disadvantage of centralized management is that the entire network may get out of control after the failure of a single manager. Compared to distributed management, centralized management may also be less efficient: it is likely that more management information needs to be exchanged and the central managers may become performance bottlenecks.

2.2 Traditional Network Management

The *Simple Network Management Protocol* (SNMP) [28] was developed in the late 1980's to provide network operators with a simple tool they could use to manage their networks. It has gained widespread acceptance since 1993, making it a standard to manage TCP/IP networks.

SNMP is based on the client-server centralized paradigm, where a central station collects and analyzes data retrieved from physically distributed network elements. The SNMP Manager makes the connections to an SNMP Agent which runs on a remote network device, and serves information to the manager regarding the device's status. The database, controlled by the SNMP agent, is referred to as the SNMP Management Information Base (MIB), and is a standard set of statistical and control values. Directives, issued by the network manager to an SNMP agent, consist of the identifiers of SNMP variables (referred to as MIB object identifiers or MIB variables) along with instructions to either GET the value for identifier, or SET the identifier to a new value. Through the use of private MIB variables, SNMP agents can be tailored for a lot of specific devices, such as network bridges, gateways, and routers. The definitions of MIB variables supported by a particular agent are incorporated in descriptor files.

The popularity of SNMP is due to a number of features. It can cover a large range of devices to be managed, and it is a very flexible and extensible management protocol. It is also proved to be good under poor network conditions. However, SNMP is not a particularly efficient protocol. Bandwidth is wasted with needless information, such as the SNMP version (transmitted in every SNMP message) and multiple length and data descriptors scattered throughout each message.

The network management systems based on Client/Server paradigm normally requires transferring large amounts of management data between the manager and agents. The large amount of data not only requires considerable bandwidth, but also can cause a processing bottleneck at the management. As current networks grow larger and more complicated, the problem becomes more severe.

The *Management by Delegation* (MbD) model was proposed in 1991 to address the difficult to manage centralized systems. The key idea of the MbD approach is to delegate management functions to remote devices in order to reduce communication costs, to avoid a single point of failure, and to increase the scalability of management applications. The management architecture of MbD still includes a management protocol and agents, yet an elastic process run-time support is assumed on each device. Instead of exchanging simple messages, the management station can specify a task by packing a set of commands to agents into a program and send it to the devices involved, thus delegating the actual execution of the task to them. This execution is completely asynchronous, enabling the management station to perform other tasks in the meantime and introducing a higher degree of parallelism in the management architecture. Moreover, since the code fragments are not statically bound to devices, they can be changed and re-sent by the management station at any time. This enables more flexibility, because the management station can customize and enhance dynamically the services provided by the agents on the devices.

Remote Monitoring (RMON) assumes the availability of network monitoring devices called monitors or probes. By monitoring packet traffic and analyzing the headers, probes provide information about links, connections among stations, traffic patterns, and status of network nodes. Hence, RMON can be regarded as traffic-oriented approach because the status of the network is determined by direct inspection of the packets flowing in it, rather than inspection of the status of each device. A probe in RMON can detect failures, misbehaviors, and identify complex relevant events even when not in contact with the management station, which is likely to happen when the network is overloaded or in critical conditions. In addition, the agent on the probe can also do periodic checking and semantic compression, which further increases decentralization.

Another solution for the problem of centralized management is the use of *Mobile Agent* (MA) technology to distribute and delegate management tasks. The emergence of mobile agent frameworks has led many researchers to examine their applicability to network management and control environments. It is believed that mobile agents can provide better solutions to performance and fault management problems, given the large amount of data that needs to be transferred in respective solutions based on traditional approaches[15]. Ten Mobile agent frameworks are currently addressed by two standards bodies. The *Federation of Intelligent Physical Agent* (FIPA) looks at high-level semantically rich interactions between software agents that deploy some form of intelligent adaptability. It has its roots in Distributed Artificial Intelligence (DAI). OMG looks mostly at the issue of mobility according to a standard interoperable framework through its *Mobile Agent System Interoperability Facility* (MASIF) [4]. In the latter, the agent systems model the execution environment able to host mobile agents.

2.3 Sensor Network Management

As described in section 1.1.2, network management becomes more and more necessary with the development of applications running on WSNs. When the requirement of management in WSNs first arose, the most natural way to do it was to try to apply what we have already had for traditional wired networks into WSNs, such as the traditional standard network management protocols, SNMP. Before we try to apply those rules or methods for designing management system for wired networks, the first question listed in section 1.3 should be answered, i.e., *"Could those rules for designing a management system for traditional wired networks be applied directly to design a SNMS?"*

The answer is “No”, because the unique challenges posed by WSNs for network management make traditional network management techniques impractical. For example, the following characteristics of wireless sensor networks, which really matter to the design of network management system, make SNMP not applicable to WSNs:

First, there is no address for each sensor, and specifying sensors is difficult. The only way to transmit messages among WSNs is to broadcast the message to all sensor nodes no matter the message is planned to be sent to all sensor nodes or only one specific node. The communication overhead becomes too high when applying SNMP directly to WSNs. Second, for some self-configured WSNs, the management server does not have all information of sensor nodes. In order to apply SNMP directly, it requires each sensor node to maintain a MIB, and the big size of MIB makes it impractical for the storage-constrained sensor nodes. Third, due to the high density of the deployment of sensor nodes, sensor-specific failures become very common. This is a unique characteristic of WSNs, and is not handled by SNMP.

Ad Hoc Network Management Protocol **ANMP**[11] and **Guerilla**[38] are two protocols designed for managing mobile wireless ad-hoc networks, but they can be used with certain types of WSNs. **ANMP** uses hierarchical clustering of nodes to reduce the number of messages exchanged between the manager and the agents. It is an extended SNMP with the differences including MIB extensions, dynamic configuration of agents, dynamic extension of the agents, and an application-specific security module. The main contribution of ANMP is to make SNMP work for wireless networks. **Guerilla** is another adaptive management architecture for ad hoc networks, which provides management flexibility and continuity by making its nomadic managers adapt to dynamic network conditions. It employs a two-tier infrastructure: the higher tier consists of groups of peer-to-peer nomadic managers that process management intelligence, adapt to network dynamics, collaborate among one another; the lower tier consists of active probes that may be dispatched to remote nodes to perform localized management operations. The nomadic managers and active probes facilitate disconnected management operations and reduce consumption of wireless bandwidth.

*Management Architecture for Wireless Sensor Networks***MANNA** [36], is a management solution specific for WSNs, but it adopts ad hoc network management techniques. It provides a general framework for policy-based management of sensor networks. It collects dynamic management information, maps this into WSN models, and executes management functions and services based on WSN models. MANNAs management policy specifies management functions that should be executed if certain network conditions are met. WSN models maintain the information about the state of the network. MANNA defines the relationship among WSN models in a Management Information Base (MIB). MANNA adapts to dynamic WSN behaviors by analyzing and updating the MIB. MIB update is a centralized operation and expensive in terms of energy consumption. Moreover, WSN uncertainties and delay may affect the accuracy of collected management information. To keep the MIB up-to-date, it is critical to determine the right time to query for management information and the right frequency for obtaining management information.

Another system based on traditional network management systems is **BOSS**[41]. It proposes a service discovery management architecture for WSNs. The architecture is based on UPnP, the standard service discovery protocol for network management. To make UPnP run on resource-constrained sensor nodes Song et al [41]. implements an UPnP agent in the base station, called *Bridge Of the SensorS* (BOSS), which provides a bridge between a managed sensor network and a UPnP network. The proposed system consists of three main components: UPnP control point, BOSS, and non-UPnP sensor nodes. The control point is a powerful logical device with sufficient resources to run the UPnP protocol and manage a sensor network using the services provided by BOSS, e.g. PCs, PDAs, and notebooks. BOSS is a base node that acts as the mediator between non-UPnP sensor nodes and UPnP control point and is implemented in the base station. Each node in a sensor network is a non-UPnP device with limited resources and sensing capability. The base node carries the network management computation burden, rather than the resource-constrained sensor nodes. The control point can specify which events of non-UPnP sensors it is interested in.

A management framework called *Sensor Network Management Protocol*, **sNMP** [12], is proposed by Deb et al. The sNMP framework has two functions: First, it defines sensor models that represent the current state of the network and defines various network management functions. Second, it provides algorithms and tools for retrieving network state through the execution of the network management functions. Models for sensors include network topology (node connectivity), energy map (node battery power), and usage patterns. Deb et al. suggest that sensor models could be used for different network management functions. The human manager could use the current knowledge of network topology for future node deployment. By measuring network states periodically, the human manager can monitor and maintain the network by identifying which parts of the network have a low performance, and taking corrective actions as necessary. From periodic monitoring of network states, the human manager could also analyze network dynamics to predict network failures and then take preventive actions.

Louis Lee et al. [31] propose an adaptive policy-based management system for WSNs, called *Wireless Sensor Network Management System* (**WinMS**). The end user predefines management parameter thresholds on sensor nodes that are used as event triggers, and specifies management tasks to be executed when the events occur. A local network management scheme provides autonomy to individual sensor nodes to perform management functions according to their neighborhood network state, such as topology changes and event detections. The central network management scheme uses the central manager with a global knowledge of the network to execute corrective and preventive management maintenance. The central manager maintains an MIB that stores WSN models that represent network states. The central manager analyzes the correlation among WSN models to detect interesting events such as areas of weak network health, possible network partition, noisy areas, and areas of rapid data changes. An advantage of WinMS is that its lightweight

TDMA (Time Division Multiple Access) protocol provides energy-efficient management, data transport and local repair. Its systematic resource transfer function allows non-uniform and reactive sensing in different parts of a network, and it provides automatic self-configuration and self-stabilization both locally and globally by allowing the network to adapt to current network conditions without human intervention. A disadvantage of WinMS is that the initial setup cost for building a data gathering tree and node schedule is proportional to network density.

Tolle and Culler [45] propose *Sensor Network Management System* **SNMS**. It is an interactive system for monitoring the health of sensor networks. SNMS provides two main management functions: query-based network health data collection and event logging. The query system allows the user to collect and monitor physical parameters of the node environment. The event-driven logging system allows the user to set event parameters and nodes in the network will report their data if they meet the specified event thresholds. The main advantage of SNMS is that it introduces overhead only for human queries and so has minimal impact on memory and network traffic. SNMS further minimizes energy consumption by bundling the results of multiple queries into a single message instead of returning results individually. The main drawbacks of SNMS are that the network management function is limited to passive monitoring only, requiring human managers to submit queries and perform post-mortem analysis of management data. Furthermore, SNMPs centralized-processing approach requires continuous polling of network health data from managed nodes to the base station, and this can burden sensor nodes that should minimize transmissions in order to extend network lifetime.

2.4 Principles of Sensor Network Management Development

In section 2.3, the reason why the traditional network management mechanism cannot be applied directly to WSNs is analyzed. To continue to seek the answer to the first question listed in section 1.3, the rules or principles of developing a Sensor Network Management System (SNMS) should be provided.

In general, the design of a SNMS can be divided into two categories: Define the SNMS framework and define the supporting algorithms and tools[7]. Defining the SNMS framework includes defining various network management functions (which may be required for sensor networks) and designing a management architecture (which takes into account specific characteristics of WSNs). Some specific characteristics of WSNs involve restrictive physical resources (such as energy and computing power), frequent reconfiguration and adaptation, and faults caused by nodes. The designing of supporting algorithms and tools needs to provide different algorithms and tools for retrieving network state and maintenance of network using the network management functions. It is mostly depending on the applications that SNMS will manage.

Since the SNMS framework will mainly decide what kind of supporting algorithms and tools that SNMS needs to perform management functions, the design of SNMS framework becomes the key part of the whole SNMS design. A good design of SNMS framework should easy the management tasks and simplify the requirements for the supporting algorithms and tools.

The design of SNMS should follow the principles listed below, which take into account the unique attributes of WSNs:

1. SNMS should be able to run with the application without wasting energy, considering the energy-constraint of sensor nodes;
2. SNMS should occupy a minimal amount of memory, considering the memory constraint of sensor nodes;
3. SNMS should generate network traffic only in response to the requests from the network manager, considering the bandwidth constraint of sensor nodes;
4. SNMS should depend on the application as little as possible, and should not impose any semantics of its own at the same time;

5. SNMS should not interfere with the operation of the application, i.e., should balance its overhead with application's performance;

6. SNMS should be simple and robust;

These principles will guide the design procedure of SNMS, and help the designer to reach the final goal of designing a lightweight SNMS.

CHAPTER THREE

SENSOR NETWORK MANAGEMENT SYSTEM (SNMS) DESIGN

3.1 SNMS Architecture

Similar to traditional wired networks, there are also three main options for the architecture of Sensor Network Management Systems: centralized, distributed, or hierarchical [10].

In centralized management systems, the base station acts as the manager that collects information from all sensor nodes and controls the entire network. The base station (the central server) with unlimited resources can perform complex management tasks, reducing the processing burden on resource-constrained sensor nodes. Since the base station also has the global knowledge of the network, it can provide accurate management decisions. But, this approach has some problems. First, it incurs a high message overhead (bandwidth and energy) from data polling, and this limits its scalability. Second, the central server is a single point of data traffic concentration and potential failure.

There are multiple management stations in distributed management systems. Each station controls a subnetwork and may communicate directly with other stations in order to perform management functions together. Distributed management has lower communication costs than centralized management, so it provides better efficiency in terms of energy. But the cooperation needed between multiple stations makes the management system more complex and difficult compared to the centralized management system. The distributed management algorithms may be computationally too expensive for resource-constrained sensor network nodes. Another disadvantage of distributed systems is memory costs. It usually requires the sensor node to store extra information about the subnetwork it belongs to, such as neighborhood state transition diagram, which is maintained in TP [19]. It requires significant memory resources.

A mobile agent-based framework is an example of distributed management system implementation [47], [2]. Some special nodes (agents) are added to the network. The central manager performs management tasks or retrieve network states of sensor nodes through these agent nodes. The main advantage of this approach is that local processing reduces network bandwidth requirements and prevents network bottlenecks by reducing processing at the central server. Furthermore, agents can be designed to distribute tasks in the network. For example, agents can relay some tasks from overloaded nodes to other nodes with lower workloads. There are several drawbacks of agent-based approaches. First, this approach requires a network to be configured manually in order to cover all nodes in the network. Second, the agent-based approach introduces delays because the central manager needs to wait for an agent to visit the node first. Third, the agent-based approach does not scale for large WSNs because as the number of sensor nodes increases, the number of agents deployed must be increased. Finally, since the agent typically sends aggregated management information from a set of managed nodes in a network, fine-grained information from individual nodes is compromised.

Hierarchical network management is a hybrid between the centralized and distributed approach. Intermediate managers are used to distribute management functions, but do not communicate with each other directly. Each manager is responsible for managing the nodes in its subnetwork. It passes information from its subnetwork to its higher-level manager, and also disseminates management functions received from the higher-level manager to its sub-network. Some common-nodes are selectively elected as cluster heads to act as distributed managers. There is a non-trivial energy overhead for selecting cluster heads.

To follow the design principles described in section 2.4, avoid the expensive computation, high memory cost in distributed network management, and architecture complexity in hierarchical network management, this SNMS is designed to be in the form of improved centralized architecture, which is shown in Figure 3.1.

The same as traditional centralized architecture, this SNMS consists of two parts: the SNMS

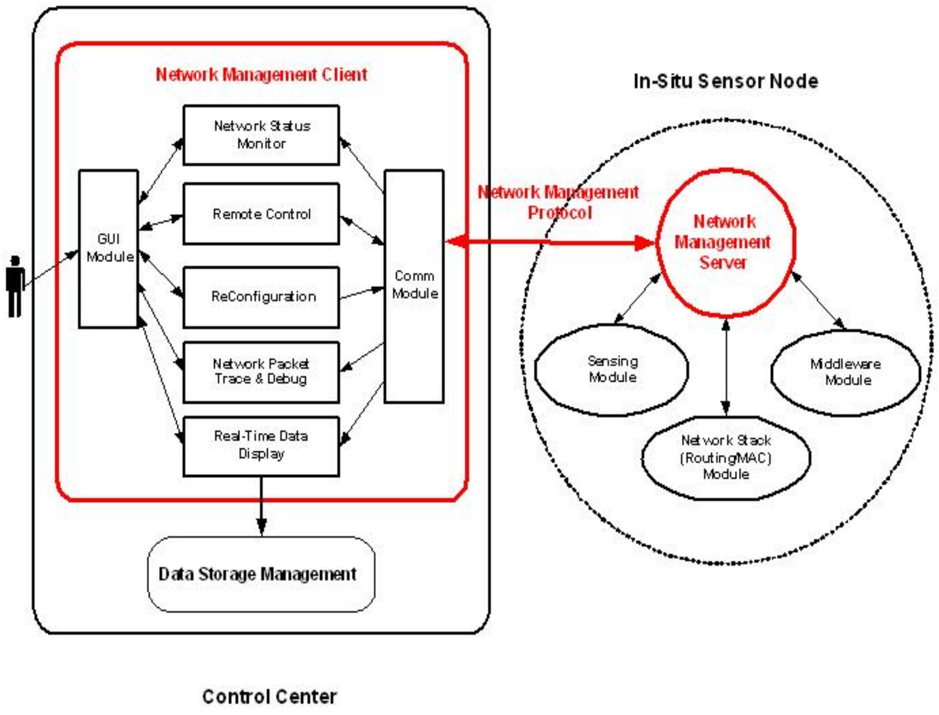


Figure 3.1: SNMS Architecure

server, which is located in sensor nodes, and the SNMS client, which is running on a PC in the control center. The client/server based centralized architecture in this design is different in the following ways:

- **Multi-server, and single-client**

  In this design, each node is a SNMS server to provide information about current network status to its client, and the base station (a PC in the control center) is the only client who requests information from sensor nodes or sends control command to adjust the performance of sensor nodes.

- **Simple-server, and Powerful-client**

  It is a simple-server and powerful-client architecture because most operation are executed on the client (base station, PC) to reduce the burden expected of the server (sensor nodes). The PC client bears the full burden of managing all information of the application needed in

SNMS, and serializing network transmissions. The sensors only need to provide the minimum support for SNMS.

3.2 SNMS Functionality

The second goal listed in section 1.3 is to define the network management functions according to the requirements of sensor networks.

Usually, wireless sensor networks are application-oriented. Different applications have different requirements for sensor network management functions [49]. Those requirements includes:

- Power management: power ON/OFF, power exhausting
- Reliability management: fault/reliability management
- Topology management: self-configuration
- Security management: security, privacy
- Traffic management: prevent congestion at or near base stations
- Context management: context-awareness management, task management, location-based context management

Most previous sensor network management systems are a set or subset of these functions based on the requirements of the application it manages, which are decided during the system design phase; different management systems have different focuses and usually are not compatible with each other after they are implemented. When a new application needs to be managed, a corresponding management system has to be designed and implemented specific for it.

The idea of the solution for this problem is to provide a generic support for network management, and make it customizable and extensible for different applications, but no change needed for the basic SNMS mechanism when applying it to different applications.

As shown in Figure 3.1, the most commonly required functions of different applications are included in this SNMS architecture, including the performance monitor and control, configuration management, fault management, and data management. All of these management modules are designed to be non-application-oriented, and provide generic support for SNMS services. For example, in performance management module, it supports remote inquiry for all network performance related parameters in an application. When it is applied to different applications, the network manager can simply select which parameters are more important for the managed application, and customize the SNMS to include the management for these parameters.

The design details for each management module will be described in the following sections in this Chapter.

3.3 Lightweight Framework

"How to cooperate the management system with the application modules to limit the management overhead and get maximized overall performance?" It is an extremely critical important concern for management systems designed for wireless sensor networks. One of our goal, also the most important one, is try to design a lightweight framework of SNMS (as mentioned in section 1.3).

Unlike the traditional wired networks, sensor nodes in WSNs generally operate with very tight resources, such as limited battery power, limited storage capacity, and constrained wireless communication. This is one of the unique attributes of WSNs.

Several schemes or mechanisms about how to minimize the overhead of SNMS, in terms of network resources consumption and the development workload, are added to our SNMS during the design process:

- **0-MIB in SNMS server**

  In the traditional network management design, for a centralized SNMS to do the network management, both SNMS client and server need to maintain a local database of information relevant to network management, known as the management information base (MIB).

MIB contains definitions and information about the properties of managed resources and the services that the SNMS server supports. The information include the current configuration, operation statistics, and parameters to be controlled of the managed resource. The SNMS server also needs to implement some packet types and a Get/Set function on its MIB variables. The MIB at the SNMS client contains network management information extracted from the MIBs of all the managed entities (sensor nodes) in the network. It is the responsibility of the SNMS server to store and update MIB, receive control packets and do the computation to perform the management action.

In this design, there is no MIB needed (called 0-MIB) in SNMS server (the sensor node). All the management information are abstracted at compile time, and stored in the SNMS client (the powerful PC). Only a small portion of code (which will handles the control packets) will be stored in each node (that meets the requirement of SNMS design for WSNs in section 2.4, i.e., try to occupy a minimal amount of memory).

It also minimizes the computation overhead for MIB management on sensor nodes. There is no complex management mechanism for managed parameters needed in sensor nodes. To respond to the inquiry from SNMS client, the sensor nodes simply get the value of inquired parameter directly based on the memory address provided in the inquiry message. No computation needed in the whole response process. This mechanism makes the support for updating of MIB in SNMS client very simple on sensor nodes, which meets the requirement of keeping the SNMS on the sensor nodes as simple as possible, and leaves all the complexity to the powerful SNMS client (a PC).

- **0-Network stack for SNMS module**

  To support management function provided by SNMS, two transmission patterns are needed: Collection and Dissemination. Collection is required to obtain management information from the network, and Dissemination is required to distribute management commands and queries.

Considering the Collection is such a typical traffic pattern for almost all applications running on WSNs, this design allows SNMS to use the application's networking stack instead of containing its own networking stack that runs in parallel with the application's. In this way, it would require less RAM, less code in the resource-limited sensor node, and would prevent redundant network maintenance traffic too.

For the Dissemination, we can choose one of the reliable broadcast protocols and combine it into current networking stack, as a special traffic case handled by the application's routing module. It can still take advantage of current networking structures in sensor nodes, instead of creating a complete separate one.

- **Runtime configurable periodical message**

  It is a common process in sensor networks to periodically export interesting parameters. For example, a collection tree component can periodically send a specific debug message, which contains the current parent, current link estimates, and current path costs. This data can be used to graph a network in realtime and study its stability. Usually, this kind of debug message can only be enabled at compile time, and once enabled, is constantly sent at a fixed period. From a developer's point of view, it is not flexible enough. Think about in the severe situation, when the application may need more bandwidth to gather more data, the overhead of this periodical management traffic will not be ignorable.

  An event-driven report system is incorporated into this SNMS design. There are two types of event messages supported by this report system: one is the one-time urgent event, which is used to report any fault happened on the sensor node; another type is the periodical event message. The difference between this periodical message and the previous debug message is, the report rate of this periodical event message is configurable during runtime. It even can be turned off when the traffic in the network is really high. This infrastructure places management at the user's control, not at the compiler's. It provides user flexibility to extract system status with different levels of detail, and alleviates the overhead of application activities at the same time.

- **Minimal effort for using SNMS**

  As discussed in section 3.2, wireless sensor networks are usually application-oriented. Different applications have different requirements for sensor network management functions. To simplify the development of SNMS for WSNs, a generic framework is provided in this SNMS design. Besides universality, another attractive advantage of this design is, only minimal effort is needed to customize the generic SNMS for a specific application. The user only needs to mark the commands on the SNMS server which he/she wants to exposed to the SNMS client. There is no extra code, specific interface, or fixed message types needed from the user to support SNMS service. All the complex computation of management functions is done in SNMS client, and add minimal overload to the SNMS server.

3.4 Main Module Design

3.4.1 Performance Management

The goal of performance management is to measure and make available various aspects of network performance so that the network performance can be maintained at an acceptable level. Performance management involves three main steps. First, performance data is gathered on variables of interest to network administrators. Second, the data is analyzed to determine normal (baseline) levels. Finally, appropriate performance thresholds are determined for each important variable so that exceeding these thresholds indicates a network problem worthy of attention, and corresponding adjustments might be sent out to the network.

As described before, due to application-specific characteristic, the performance management is much different in WSNs. There are a wide range of applications for sensor networks with different requirements [39],[8],[43]. Different applications have different performance metrics, such as identification accuracy, probability of loss-of-track, the status of the battery power (expected remaining lifetime), link quality, and location (longitude and latitude). All this information might need to be collected as (and when) it changes significantly.

The idea behind the design of our performance management module is to try to provide a mechanism to abstract the management related information all at once, and then form a generic information base. It is non-application-specific, that means different applications can base on it to form their own information bases, and perform the network management through the same mechanism. No change needed in SNMS when it is applied to different applications.

After the construction of the generic information base, the manager in control center, has the full flexibility to decide which performance metrics need to be calculated and monitored for this specific application. The computation and data process are the same as it is designed dedicated for a specific application. The only difference (also the advantage of using this generic information base) is the user can inquiry any variable of the application directly through the information base, no pre-defined interface needed for each specific variable.

The underlying method beneath this mechanism is called Remote Procedure Call (RPC)[6],[5],[46]. It allows a PC to access the functions and variables of the statically-compiled program on a wireless embedded device at run-time. The client provides the equivalent of a remote terminal to an embedded device: the network operator opens an interpreter on the PC and is presented with a set of objects representing the software modules actually running on the node. Through these objects, the node's functions can be called, its variables on the node's heap can be read and written, and its enumerations and data structures can be accessed. With RPC, embedded applications seamlessly span the PC and the sensor node.

No extra code must be written by the developer to use RPC; a minimal set of hooks is automatically added to the application at compile time. The PC client software imports all information from the application at compile-time. In contrast to batch programming in which the network program runs autonomously, RPC allows the developer to observe or change the state of a node at a runtime, and to compose the functions of a node application in new ways without downloading any new code to the node.

In order to get or set variables on the heap of sensor nodes through RPC, when the embedded application is compiled, the name and type of each variable declaration is extracted from the source code, and the memory address on the heap is extracted from the symbol table of the executable. The PC client is equipped with a mechanism that provides the ability to read and write directly any memory address. Most of the work is done by the PC client, which reads the variable sizes and memory addresses abstracted at the compiling time, and calls pre-created set/get functions on the node using the RPC system described earlier. Because the client knows the type of the variable, it is responsible for indexing into arrays, dereferencing pointers, and casting return arguments to the appropriate type.

Supported by this mechanism, a sensor network management system can perform a variety of management control tasks based on the abstracted network states. For example, the network management system can provide remote power management by get/set a variable on the node's heap, called $powerMode$. When the user sends out a command through RPC to place a set of nodes into a fully-awake state by setting $powerMode$ to AWAKE; or to turn them to a sleep state or a hibernation mode, by set $powerMode$ to different values. The program on the node will check on the current value of $powerMode$ and adjust the status of node accordingly. Another example is to monitor the battery voltage which is a variable in the heap of sensor node to predict node failure.

3.4.2 Fault Management

The goal of fault management is to detect, log, notify users of, and (to the extent possible) automatically fix network problems to keep the network running effectively. Because faults can cause downtime or unacceptable network degradation, fault management is perhaps the most widely implemented of network management elements [37].

Compared to wired networks, the sensor nodes in WSNs always work in a harsh environments, the failure happens much often than nodes in wired networks. So efficiently detecting the faults and making the sensor nodes self-healing in most cases become more important for WSNs. The fault management in this SNMS provides a set of mechanisms to handle different types of faults that could happen in WSNs.

- **Event-driven fault logging system**

  An important fault management function SNMS provided is event logging. It can be used to record any unusual event for post-analysis and real-time monitoring of unexpected events.

  Event messages generated on a running mote will be reported to SNMS module, and then be delivered to base station by radio. In the base station, all event messages will be logged into persistent local storage.

  To make the program interface generic, the event logging system does not embed meaning within the messages, nor does it interpret them in any way. Instead, every event is represented by a programmer-created string, which is intended to be meaningful to a human manager of the system. This string is associated with a set of values to be captured from variables at runtime. This aspect of the design was inspired by the UNIX “syslog” facility, which has the benefit of being time-tested.

  The event logging in this SNMS is designed to be configurable. The manager at control center can send a remote command using the command dissemination layer to set filter/trigger for event to bound the network bandwidth consumed by event logging. By default, no periodical event is allowed considering the communication constrain of sensor nodes. The alert events are only triggered by sensor node modules in some urgent situation.

- **Software fault recovery mechanism**

  For the software failures, SNMS provides a protection using WatchDog timer which is supported by the operating system on sensor motes [29]. This timer forcibly reboots the mote if the application does not periodically reset a flag.

3.4.3 Configuration Management

The goal of configuration management is to monitor network and system configuration information so that the effects on network operation of various versions of hardware and software elements can be tracked and managed. Configuration management involves initialization and shutdown of the network. It also involves maintaining, adding, and updating new network components [9].

Recall that wireless sensor networks represent a new computing class consisting of large numbers of highly resource-constrained nodes which are often embedded in their operating environments, distributed over wide geographic areas, or located in remote regions. These networks must operate unattended for extended periods of time during which evolving analysis and requirements can change application semantics, creating the need to alter system behavior. Many such changes are possible by varying management parameters, executing database queries, or downloading scripts. More substantial changes require installing new program binaries using single- or multi-hop wireless network programming schemes[27].

Network programming provides the ability to wirelessly install a new program image. This is accomplished by propagating a program binary over the wireless network and having each node program themselves with the new image. It provides great flexibility and convenience for retasking large-scale, embedded, distributed, and remote systems.

The SNMS provides the network programming by embedding Deluge 2.0 [20] into SNMS module.

CHAPTER FOUR

SNMS IMPLEMENTATION

4.1 Development Environment

4.1.1 Software Development Environment

To implement the SNMS design proposed in chapter 3, two parts need to be developed separately: the SNMS server and the SNMS client. Compared with the SNMS client, which will be developed in high-level OOP language, Java, and run on powerful PC, the implementation of the SNMS server is not that easy because of the critical embedded operating system in which it will be developed, and the special properties of the developing language it uses.

As we know, the traditional tasks of an operating system are controlling and protecting the access to resources and managing their allocation to different users as well as the support for concurrent execution of several processes and communication between these processes [44]. These tasks are, however, only partially required in an embedded system as the executing code is much more restricted and usually much better harmonized than in a general-purpose system.

Rather, an operating system or an execution environment for WSNs should support the specific needs of these systems. In particular, the need for supporting concurrency, which is crucial for WSN nodes, as they have to handle data communing from arbitrary sources at arbitrary points in time. A simple, sequential programming model is clearly insufficient, and a event-based programming model is needed. Another need for energy-efficient execution requires support for energy management. Also, external components - sensors, the radio modem, or timers - should be handled easily and efficiently. All of these require an appropriate programming model, a clear way to structure a protocol stack, and explicit support for energy management - without imposing too heavy a burden on scarce system resources like memory or execution time.

The operating system **TinyOS** [18], along with the programming language **nesC**[16], meet the above requirements of such embedded system like sensor node.

TinyOS supports modularity and event-based programming by the concept of components. A *component* contains semantically related functionality, for example, for handling a radio interface or for computing routes. Such a component comprises the required state information in a *frame*, the program code for normal tasks, and handlers for events and commands. Both events and commands are exchanged between different components. Components are arranged hierarchically, from lower-level components close to the hardware to high-level components making up the actual application. *Events* originate in the hardware and pass upward from low-level to high-level components; *commands*, on the other hand, are passed from high-level to low-level components.

In TinyOS, the actual computational work is done in the *tasks*, which have to run to completion, but can be interrupted by *handlers*. The advantage is twofold: there is no need for stack management and tasks are atomic with respect to each other. Still, by virtue of being triggered by handlers, tasks are concurrent to each other. Multiple tasks can be triggered by several events and are ready to execute. It is done by a simple First In First Out (FIFO) scheduler, which shuts the node down when there is no task executing or waiting.

To make clear the feedbacks from the handlers and tasks which are required to run to completion, the split-phase programming approach is adopted, which splits invoking a request and the information about answers into two phases: the first phase is the sending of the command, the second is an explicit information about the outcome of the operation, delivered by a separate event.

To handle a large number of commands and events especially when using split-phase programming, the **nesC** language allows a programmer to define interface types that define commands and events that belong together. This allows to easily express split-phase programming style by putting commands and their corresponding completion events into the same interface. Components then provide certain interfaces to their users and in turn use other interfaces from underlying components. Furthermore, primitive components or modules can be combined into larger configurations by simply "wiring" appropriate interfaces together in nesC. For this wiring to take place, only components that have the correct interface types can be plugged together.

Using these component definition, implementation, and connection concepts, TinyOS and nesC together form a powerful and relatively easy to use basis to implement both core operating system functionalities as well as communication protocol stacks and application functions. Experience has shown [16] that in fact programmers do use these paradigms and arrive at relatively small, highly specialized components that are then combined as needed, proving the modularity claim. Also, code size and memory requirements are quite small. Overall, TinyOS can currently be regarded as the standard implementation platform for WSNs.

4.1.2 Hardware Components

The application of WSNs is usually running on a collection of wireless sensor nodes together with a base station (PC). For the application with the SNMS support, the base station will work as the control center, with the SNMS client running on it. The sensor nodes need to be selected as the SNMS server.

Not surprisingly, there is not a single, "perfect" wireless sensor node; Different application will require different trade-offs and different architectures. But to fulfill the principal tasks of a node, such as computation, storage, communication, and sensing/actuation, a basic sensor node has to include the following main components (Figure 4.1)[23]:

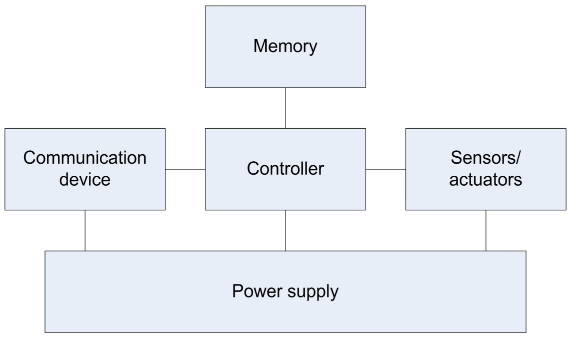


Figure 4.1: Sensor Node

- **Controller** A controller is the core of a wireless sensor node. It collects data from the sensors, processes this data, decides when and where to send it, receives data from other sensor nodes, and decides on the actuator's behavior. It has to execute various programs, ranging from time-critical signal processing and communication protocols to application programs; it is the Central Processing Unit (CPU) of the node. Such a variety of processing tasks can be performed on various controller architectures, representing trade-offs between flexibility, performance, energy efficiency, and costs. Those processors which specifically geared toward usage in embedded systems, are commonly referred as microcontrollers, such as Intel StrongARM, Texas Instruments MSP 430, and Atmel ATmega.

- **Memory** Memory is used to store programs and data; usually, Random Access Memory (RAM) is used to store intermediate sensor readings, packets from other nodes, and so on; Program code can be stored in Read-Only Memory (ROM) or , more typically, in Electrically Erasable Programmable Read-Only Memory (EEROM) or flash memory.Flash memory can also serve as intermediate storage of data in case RAM is insufficient or when the power supply of RAM should be shut down for some time.

- **Sensors and actuators** The actual interface to the physical world: devices that can observe or control physical parameters of the environment. Sensors can be roughly categorized into three categories: Passive omnidirectional sensors, Passive narrow-beam sensors, and Active sensors. Most of the theoretical work on WSNs considers passive, omnidirectional sensors. Narrow-beam-type sensors like cameras are used in some practical testbeds, but there is no real systematic investigation on how to control and schedule the movement of such sensors. Active sensors are not treated in the literature to any noticeable extent.

- **Communication** Turning nodes into a network requires a device for sending and receiving information over a wireless channel. The choices of transmission medium include radio frequencies, optical communication, and ultrasound. Radio Frequency (RF)-based communication is by far the most relevant one as it best fits the requirements of most WSN applications:

It provides relatively long range and high data rates, acceptable error rates at reasonable energy expenditure, and does not require line of sight between sender and receiver.

- **Power supply** As usually no tethered power supply is available, some form of batteries are necessary to provide energy. Sometimes, some form of recharging by obtaining energy from the environment is available as well (e.g. solar cells).

There are quite a number of actual nodes available for use in wireless sensor network research and development [17]. A few examples are the “MICA Mote” family, EYES nodes, BTnodes, and Scatterweb.

As mentioned above, when choosing the hardware components for a wireless sensor node, evidently the application’s requirements play a decisive factor with regard mostly to size, costs, and energy consumption of the nodes. For example, in some extreme cases, an entire sensor node should be smaller than 1cc, weigh (considerably) less than 100g, be substantially cheaper than US 1 dollar, and dissipate less than 100uW. In more realistic applications, the mere size of a node is not so important; rather, convenience, simple power supply, and cost are more important.

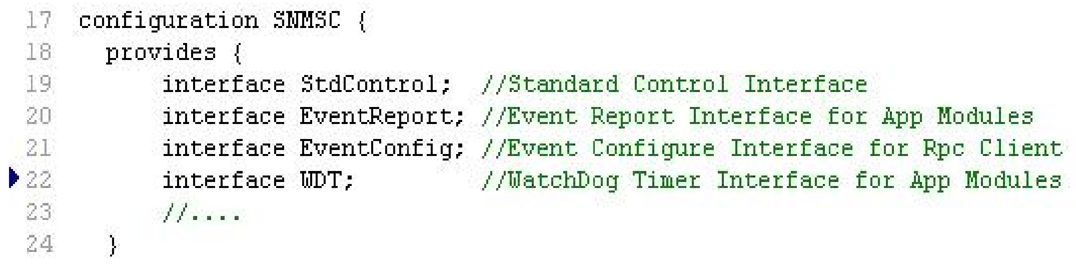
4.2 SNMS Server

In our SNMS design, SNMS server has the responsibility to provide information requested by its client, and perform the actions following the commands sent by its client. It also has the self-managing function to trigger the one-time urgent report when any fault happens on itself. Besides providing all these services and functions, the implementation of SNMS server should meet the criteria, including generic, flexible and extendable.

4.2.1 Generic Interface

In order to provide a generic framework of SNMS, which can be easily wired to other application modules, it is very important to design the interface provided by SNMS. We take advantage of

Interface design supported by nesC language, and design the interface of our SNMS, shown in Figure 4.2.



```
configuration SNMSC {
  provides {
      interface StdControl;  //Standard Control Interface
      interface EventReport; //Event Report Interface for App Modules
      interface EventConfig; //Event Configure Interface for Rpc Client
      interface WDT;         //WatchDog Timer Interface for App Modules
      //....
  }
```

Figure 4.2: Interfaces Provided by SNMS

Figure 4.3 shows the top-level configuration of an application which is wiring to SNMS.

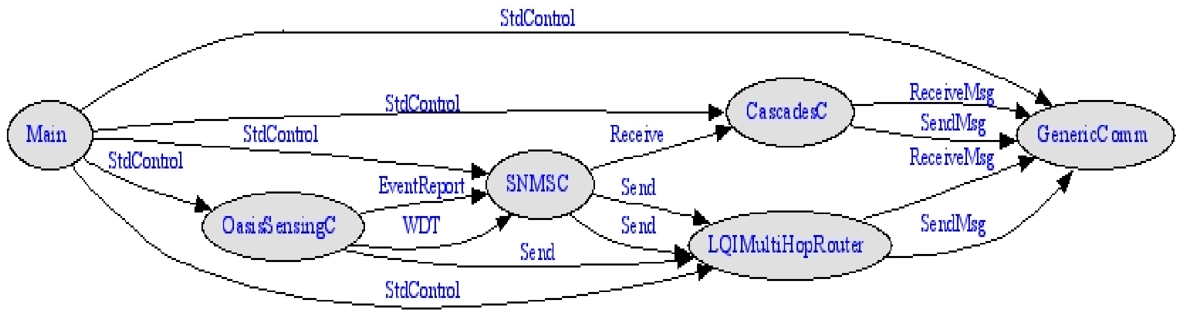


Figure 4.3: Apply SNMS to OasisSensing Application

The figure is automatically generated by a TinyOS graphic tool based on the configuration file TestOasisSNMS.nc of this application. Each circle in the figure represents a top-level component with its name marked inside it. The solid arrow line represents a connection (wiring) between two components. For example, $OasisSensing$ component is connected to $SNMSC$ component through two solid arrow lines, it means $OasisSensing$ uses the two interfaces, WDT and $EventReport$, provided by $SNMSC$ module.

The $Main$ component finishes the initialization of all components through $StdControl$ interface provided by each component. $OasisSensing$ is the main task of the sensor node. It gathers the value of environment parameters and provides them to the routing module $LQIMultiHopRouter$, which will find the path for the data to reach the sink node. $SNMSC$ component provides support for network management services to other modules. Routing module $LQIMultiHopRouter$ and $CascadesC$ provide Collection and Dissemination routing for $SNMSC$ module. $GenericComm$ component finishes the transmission of all data or control commands.

4.2.2 SNMS Server Architecture

The component architecture inside the SNMS server module is shown in Figure 4.4.

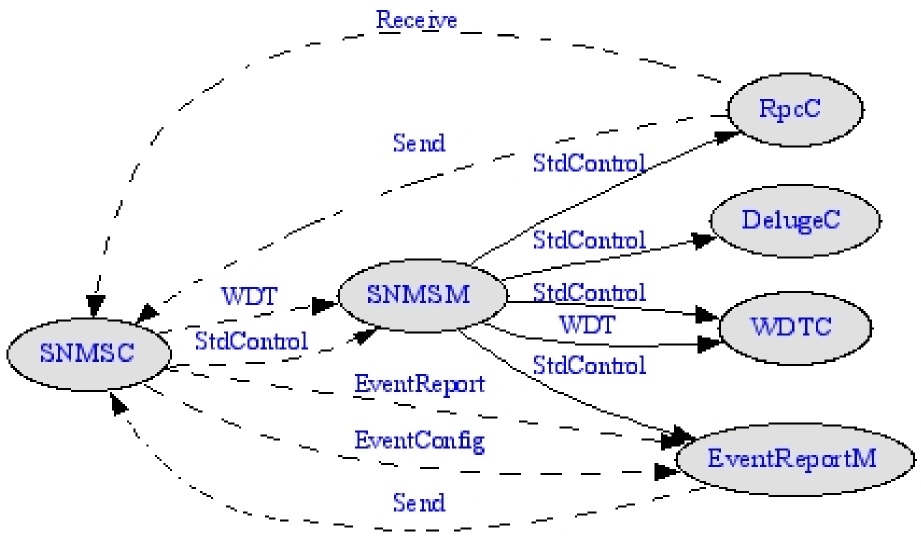


Figure 4.4: Component Architecture of SNMS Server

The following submodules are included in SNMS to management services:

- **EventReportM** module to provide event reporting mechanism
- **DelugeC** module to provide support for remote reprogramming
- **WDTC** module to provide support for WatchDog self-healing function
- **RpcC** module to provide support for remote control

The doted arrow line means the interface is not directly provided by the connected component. For example, $RpcC$ component uses two interfaces, $Send$ and $Receive$ from $SNMSC$ component. These two interfaces are not implemented in $SNMSC$ component. $SNMSC$ component is just the intermediate component to connect $RpcC$ component to the real provider of these two interfaces (that relationship or connection can be reflected in the upper level figure.)

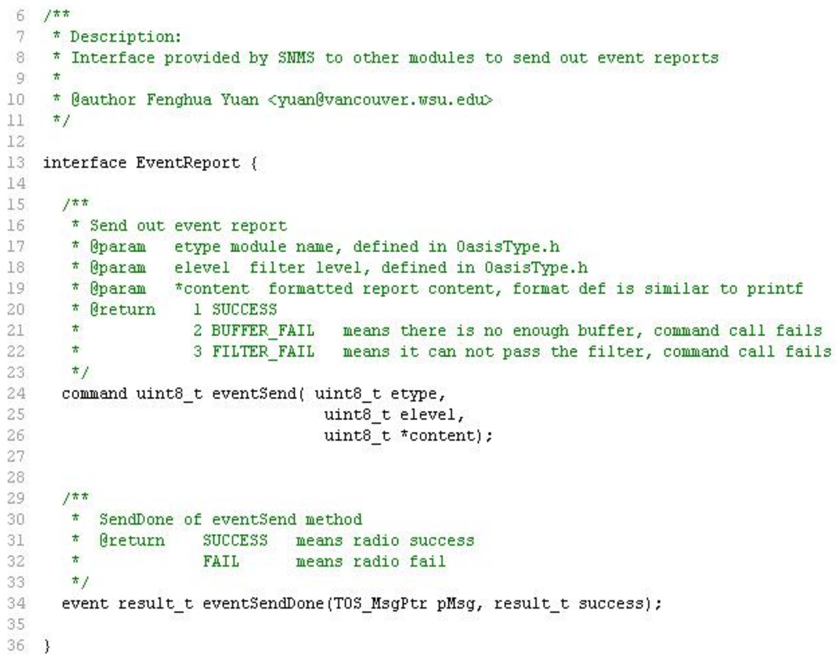
4.2.3 SubModule Implementation

Event Report Module

Event report module supports program-driven notification of on-time events. It provides a mechanism for post-mortem analysis and real-time monitoring of unexpected events. It provides two interfaces: one for other modules in the application to report event with user assigned event level (Figure 4.5); another for the SNMS client (on PC) to remotely configure the allowed event level for each module, only the event with lower level (means more urgent) than the configured filter level can be passed to the SNMS client by radio (Figure 4.6). The eventSend() command in EventReport interface supports the formatted string (similar to the format definition in printf() in C language), leaves the flexibility to the user to define and interpret the meaning of the reported events.

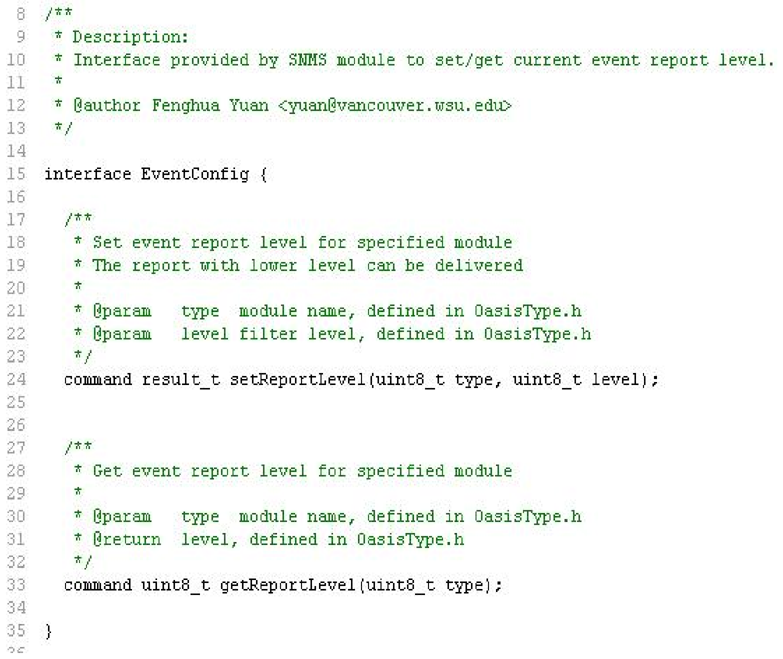
Network Programming Module

The core service required to enable network programming is the dissemination of a program image over a multihop WSN and presents several problems. First, program images are much larger than the data objects that previous dissemination protocols consider. Second, the dissemination must tolerate node densities which can vary by factors of a thousand or more. Third, complete reliability is required since every byte must be correctly received by all nodes that need reprogramming, even in the presence of high loss rate and evolving link qualities common to WSNs. Fourth, propagation must be a continuous effort to ensure that all nodes receive the newest code since network membership is not static: nodes come and go due to temporary disconnections, failure, and network repopulation. Finally, the dissemination process should require a minimal amount of



```
/**
 * Description:
 * Interface provided by SNMS to other modules to send out event reports
 *
 * @author Fenghua Yuan <yuan@vancouver.wsu.edu>
 */

interface EventReport {

  /**
   * Send out event report
   * @param   etype module name, defined in OasisType.h
   * @param   elevel  filter level, defined in OasisType.h
   * @param   *content  formatted report content, format def is similar to printf
   * @return    1 SUCCESS
   *            2 BUFFER_FAIL   means there is no enough buffer, command call fails
   *            3 FILTER_FAIL   means it can not pass the filter, command call fails
   */
  command uint8_t eventSend( uint8_t etype,
                            uint8_t elevel,
                            uint8_t *content);


  /**
   *  SendDone of eventSend method
   *  @return    SUCCESS   means radio success
   *             FAIL      means radio fail
   */
  event result_t eventSendDone(TOS_MsgPtr pMsg, result_t success);

}
```

Figure 4.5: EventReport interface provided by EventReport module



```
/**
 * Description:
 * Interface provided by SNMS module to set/get current event report level.
 *
 * @author Fenghua Yuan <yuan@vancouver.wsu.edu>
 */

interface EventConfig {

  /**
   * Set event report level for specified module
   * The report with lower level can be delivered
   *
   * @param   type  module name, defined in OasisType.h
   * @param   level filter level, defined in OasisType.h
   */
  command result_t setReportLevel(uint8_t type, uint8_t level);


  /**
   * Get event report level for specified module
   *
   * @param   type  module name, defined in OasisType.h
   * @return  level, defined in OasisType.h
   */
  command uint8_t getReportLevel(uint8_t type);

}
```

Figure 4.6: EventConfig interface provided by EventReport module

time, reducing any service interruptions to a deployed application and the debugging and testing cycle.

A number of bulk data dissemination protocols suitable for network programming have been proposed. These protocols include Multi-hop Over-the-Air Programming [42], Deluge [21], Infuse [26], and Multi-hop Network Programming [27]. The protocols differ in their design choices but they all have one thing in common that they are used to disseminate a program over a one- or multi-hop sensor network. Since Deluge is the de facto TinyOS network programming system, it is selected to be combined into SNMS to provide network programming function.

Deluge is an epidemic protocol and operates as a state-machine where each node follows a set of strictly local rules to achieve a desired global behavior: the quick, reliable dissemination of large data objects to many nodes. In its most basic form, each node occasionally advertises the most recent version of the data object it has available to whatever nodes that can hear its local broadcast. If S receives an advertisement from an older node, R, S responds with its object profile. From the object profile, R determines which portions of the data need updating and requests them from any neighbor that advertises the availability of the needed data, including S. Nodes receiving requests then broadcast any requested data. Nodes then advertise newly received data in order to propagate it further.

Deluge 2.0 [20] works with a bootloader and a TOSBoot to reprogram a node. The main steps of performing a network programming is described in the following flowchart (Figure 4.7).

The first three steps have to be done through the serial port when the sensor node connects to PC. When “inject” or “reboot” command are sent from the SNMS client, the Deluge module, which is combined in the SNMS server on sensor nodes, will responses to the command to finish the network programming.

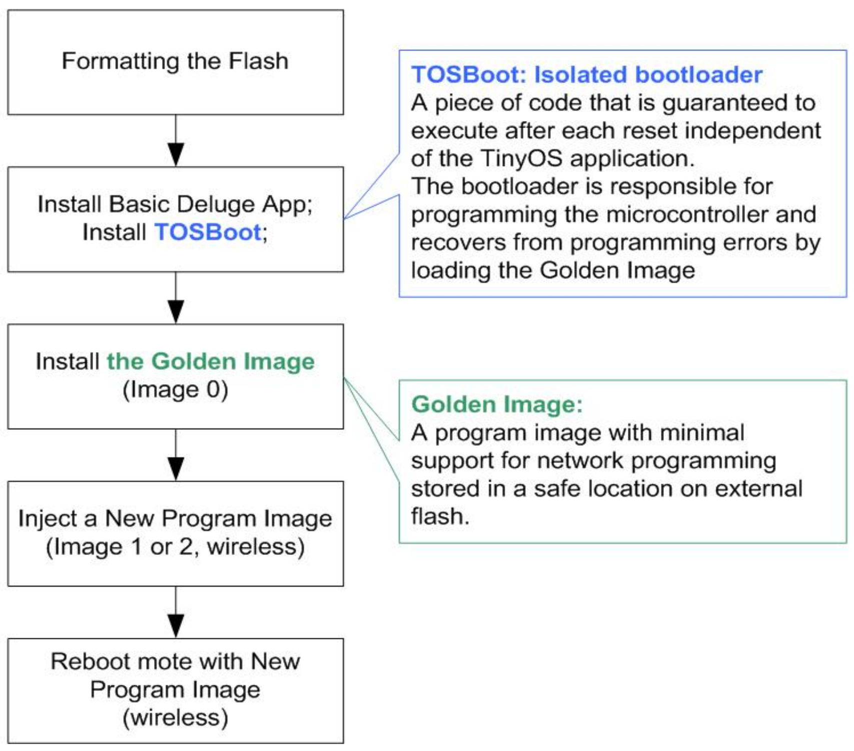

Formatting the Flash
Install Basic Deluge App; Install TOSBoot;
Install the Golden Image (Image 0)
Inject a New Program Image (Image 1 or 2, wireless)
Reboot mote with New Program Image (wireless)
TOSBoot: Isolated bootloader
A piece of code that is guaranteed to execute after each reset independent of the TinyOS application.
The bootloader is responsible for programming the microcontroller and recovers from programming errors by loading the Golden Image
Golden Image:
A program image with minimal support for network programming stored in a safe location on external flash.

Figure 4.7: Deluge: Network Programming

4.3 SNMS Client

The client of SNMS will run on a PC working as a control center, and perform management to sensor network. A Base Station (or called Sink or Root) sensor node is required to connect to the PC. This node works as the bridge between PC (client) and other sensor nodes (servers) in sensor network. All packets received by the Base Station node from other nodes through radio will be forwarded to PC through serial port, from which the Base Station connects to PC. The Base Station can also forward the packets to its TCP port. For the second case, the client can be set up at any PC which is connected to the Ethernet.

4.3.1 Communication Interface

The key part of the SNMS client implementation is how to set up the communication between the PC and the sensor network. In other words, we need a communication interface between the SNMS server and client, which are written in different languages (i.e., one in nesC, and another in

Java), and running on different platforms (i.e., one on an embedded system of sensor nodes, and another on cygwin of a PC).

As introduced in section 4.1.1, TinyOS is a component- oriented language that runs on embedded systems, mainly sensor nodes. The software written with TinyOS is built using an event-driven model and a large set of interfaces.

Java, on the other hand, is an object-oriented language that runs on a PC. Instead of interfaces, many Java applications rely on object-oriented techniques to hide the implementation by using public, private, and protected access to methods.

TinyOS typically dictates that a sensor node interacts with a Java application through a SerialForwarder object. The SerialForwarder is written to read packets from the UART, and notify any listeners of the packet with certain type. To use this object, the developer must create a class, register the packet the user is interested in, and implement the functionality to process the packet.

MoteIF (Figure 4.8) provides an application-level Java interface for receiving messages from, and sending messages to, a mote (sensor node) through a serial port, TCP connection, or some other means of connectivity. MoteIF uses buildSource to create default source, PhoenixSource, which is based on PacketSource and can automatically read and dispatch messages to all listeners using Packetizer class, which implements a mote-PC protocol.

net.tinyos.message

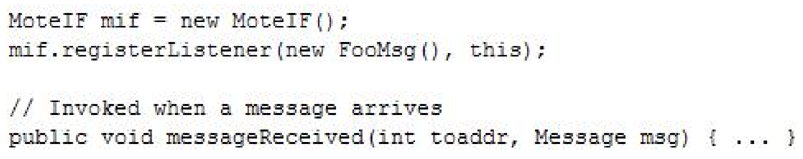
Class MoteIF

```
java.lang.Object
  └ net.tinyos.message.MoteIF
```

Figure 4.8: Class MoteIF

The default MoteIF constructor uses the MOTECOM environment variable to determine how the Java application connects to the mote. For example, a MOTECOM setting of “serial@COM1” connects to a base station using the serial port on COM1. In initialization of moteIF, it starts the source (create if needed), and create receiver and sender for source. if source is specified by host and port, moteIF will create an interface to SerialForwarder (create a SerialForwarderStub).

The default way to use MoteIF is to create an instance of this class and then register one or more MessageListener objects that will be invoked when messages arrive. For example:



```
MoteIF mif = new MoteIF();
mif.registerListener(new FooMsg(), this);

// Invoked when a message arrives
public void messageReceived(int toaddr, Message msg) { ... }
```

Figure 4.9: MoteIF Interface

Register a listener for given messages type. The message m should be an instance of a subclass of Message, generated by MIG. MIG, which comes packaged with the nesC compiler, allows a node to marshal data into a message and have it automatically unmarshaled and deserialized when received at the PC. It allows a TinyOS message type to be quickly ported to a Java class.

When a message of the corresponding type is received, a new instance of m's class is created with the received message as data. This message is then passed to the given MessageListener. Multiple MessageListeners can be registered for the same message type, and in fact each listener can use a different template type if it wishes (the only requirement is that m.getType() matches the received message).

4.3.2 SNMS GUI

The SNMS client is designed as a set of java tools which are organized in a multi-tabbed GUI framework. A moteIF object is created at the same time the GUI is created, and the MessageListeners are distributed among these tools. Each tool is triggered by the packets received through these MessageListeners and acts correspondingly, such as refresh the displayed network topology graph, update the network status calculated based on the received packets, and log the packets into record file, etc.

The software architecture of SNMS GUI is shown in Figure 4.10. More details about SNMS GUI design will be described in Chapter 5.

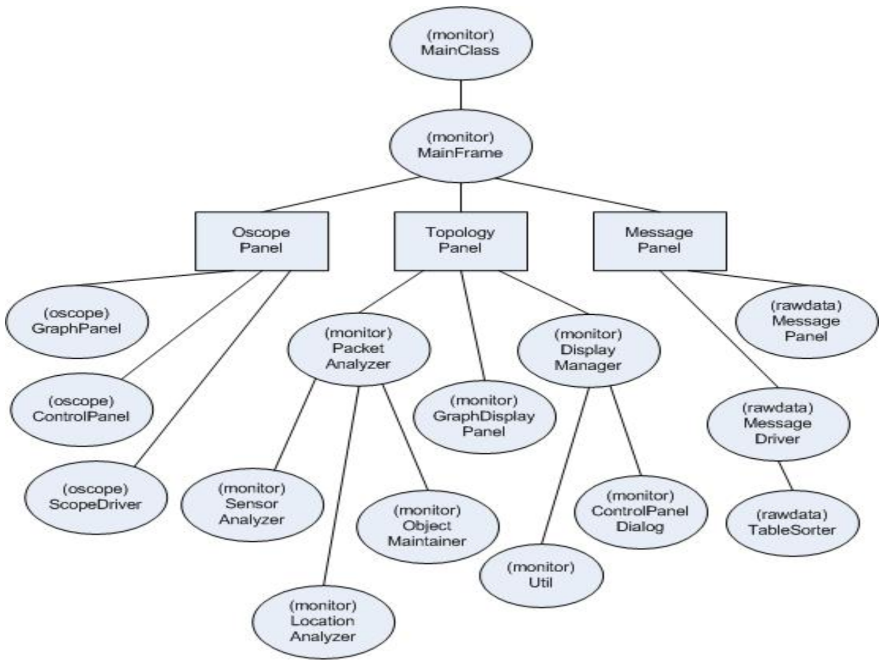


Figure 4.10: Software Architecture of SNMS client

4.4 RPC implementation

Remote procedure call (RPC) is a completely general abstraction that allows procedure calls across language, protection, and machine boundaries. Compared with the traditional RPC, the design constraints for the embedded networking domain are different:

- RPC commands need to cross machine boundaries, but not necessarily protection boundaries. Since the client and server are in the same administrative domain, they can trust each other.
- The RPC client is generally several orders of magnitude more powerful than the RPC server.
- A single client may be interacting with multiple servers, all of which will probably be programmed in the same language.

Considering the unique attributes of this kind of client/server architecture, and following the design principal of simplifying resource-constrained server and leaving complexity to the powerful client, most work is done at compile time on the PC client, and the most complicated format converting and interpreting are taken care of on the PC client too.

4.4.1 RPC Mechanism

Figure 4.11 shows the generation process of RPC during compile time, and the Remote Procedure Calls during run time. The compile time actions are supported by Tinyos 1.x, which add the RPC function stub in the SNMS server. The run time actions are implemented by our SNMS to form a run time generic MIB for the remote control. The main steps of the whole procedure will be explained in detail in the following subsections.

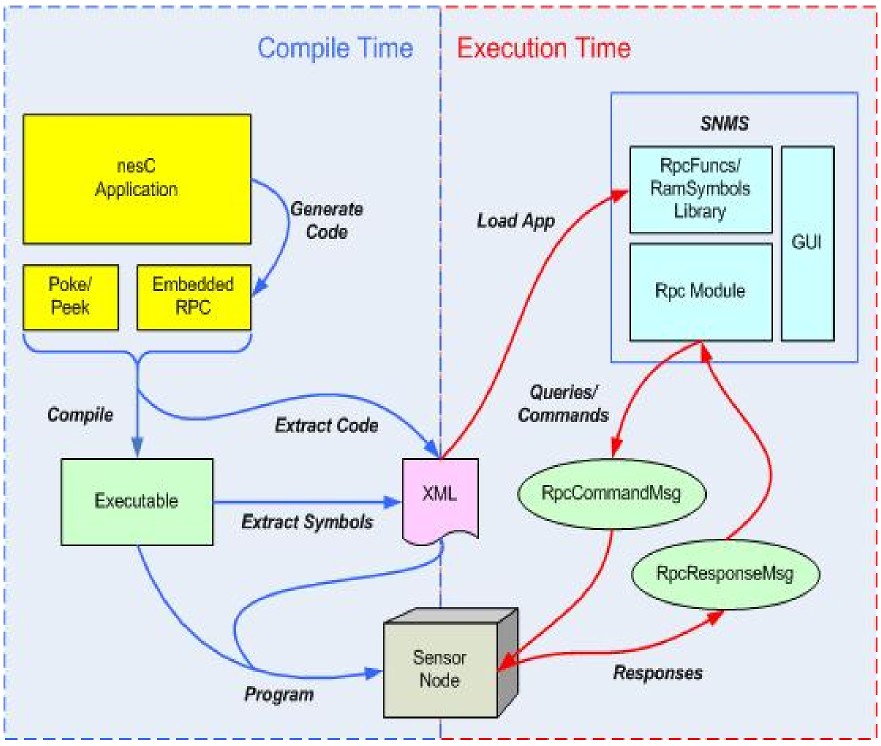


Figure 4.11: Remote Procedure Call for Remote Control

Compile Time Actions

At compile time, a RPC server stub is automatically generated and added to nesC application: the hooks for each RPC function marked with “@rpc” tag by the application designer are automatically added to the RPC module which is already wired to the application.

At the same time, all information necessary for a RPC client to use RPC is parsed from the application source code and exported to an XML file by a Perl program. All extracted nesC declarations are written to an XML file called nescDecls.xml, including enumerations, constants, data structures, typedefs, message formats, and module and interface names. The names, fields and byte alignment information of all C-structs are provided directly by the nesC compiler using tools similar to those used for MIG.

Extracting this information from the application eases the burden on the programmer, who no longer needs to repeat enumeration definitions in the RPC client or to manually perform type casting or byte alignment.

Execution time Actions

At runtime, the RPC client imports all information from the XML file. Then, the user can send commands to and receive responses from the nodes through the RPC client over a multi-hop routing layer.

Since all data is transmitted over the network using the native types of the RPC server. This adds complexity to the RPC client, which must be able to convert it to the format of CPU architecture on the PC. Besides the basic types, the RPC client also supports complex types such as arrays and structs, which use the same type alignment used on the embedded processor. The RPC client can also automatically parse nested message structures, such as when the MAC and routing protocols each add header bytes to a packet. The work done by the RPC client relieves a significant burden from the RPC server, which no longer must perform any serialization or deserialization.

Once the RPC client can generate variables and messages of the same types as those used on the sensor nodes, it reads the RPC interface definition and makes new client objects with functions identical to those on the node. These functions take nesC typed variables as arguments and, when called, pack the arguments appropriately and unpack the response messages. A similar procedure is performed for each variable on the heap. This set of objects provide a basis for user interaction that use functions and variables from the sensor nodes.

RPC calls/response

RPC requires a transport layer to pass queries and responses between the RPC client (PC) and the RPC server (network nodes).

At compile time, the user must enable RPC scripts, which parse the application code and automatically generate a number of hooks into it. Once the compiled binary is installed on a network, the user can open a RPC "terminal" into that network by specifying how to connect to the network (e.g. through the serial port). Once the terminal is open, a single object called *app* is available, through which the user can access the application installed on the nodes.

The *app* variable provides access to all software modules running on the node and their functions and variables as well as types, enumerations, and messages defined in the application. All variables of all nesC modules are available by default. Considering the cost of making all functions available, only the functions or interfaces marked with the "@rpc" tag will be imported. Marking functions and interfaces with this tag is the only effort required of the user in order to use RPC.

4.4.2 Memory Footprint

Based on the RPC mechanism, the size of our SNMS module on the sensor node is 558 bytes out of the 4k of available RAM for MICAz mote. The total size of the data gathering application (basic version of the OASIS application) that our SNMS is managing is 3257 bytes. As an important function module in the application, our SNMS only occupies a small portion (around 1/6) of total size of the application. Compared to the traditional management mechanism, our SNMS saves around 1K RAM by not storing those performance related parameters on the sensor node to form the MIB. To support RPC mechanism, only 160 bytes of RAM (out of 558 bytes) are needed for buffering packets.

At the same time, instead of adding a complex MIB management mechanism, an RPC server stub is added for each RPC function, which adds approximately 100 bytes of program memory

(ROM) for marshaling and unmarshaling the function arguments. Although ROM is not a strict constraint of most types of the sensor nodes, the code size of our SNMS on the sensor node is only 2924 bytes of ROM out of 19744 bytes, which is the total size of the testing application.

CHAPTER FIVE

SNMS DEMONSTRATION AND TESTING

Based on the work in previous chapters, a generic SNMS module was designed and implemented, which supports network status monitoring, remote control, attribute query, event logging and node reconfiguration. To test the validity of these core set of management services in our SNMS design, we configured the generic SNMS for OASIS project scenario, and wired it into OASIS data gathering application. Then make it run on a TestBed, which is setup for OASIS project.

5.1 Scenario

It is mentioned in section 1.2 that OASIS is a typical configuration for environmental monitoring and consists of one or more base stations, also called *sinks*, and a large number of wireless sensors. These sensors periodically report data to the sink. Specifically in OASIS, the different sensors equipped in a node will sense and report Seismic, Infrasonic, Lightning data to the sink in different frequency.

There are two main purposes of adding SNMS into this application: First, it provides a way for network manager to monitor the network status, and make sure it works correctly. Second, it allows geologists to adjust the focus of the sensor network to certain types of data or special spots in the volcano area if something abnormal happens.

5.2 TestBed Setup

5.2.1 Hardware Setup

An in-door TestBed is setup for the software testing in the OASIS project. It is a 22-node wireless sensor network installed on the ceiling in the VELS building on the WSU vancouver campus. Figure 5.1 and 5.2 show the layout of the sensors in the TestBed and an sample of the sensor node case in the TestBed separately.

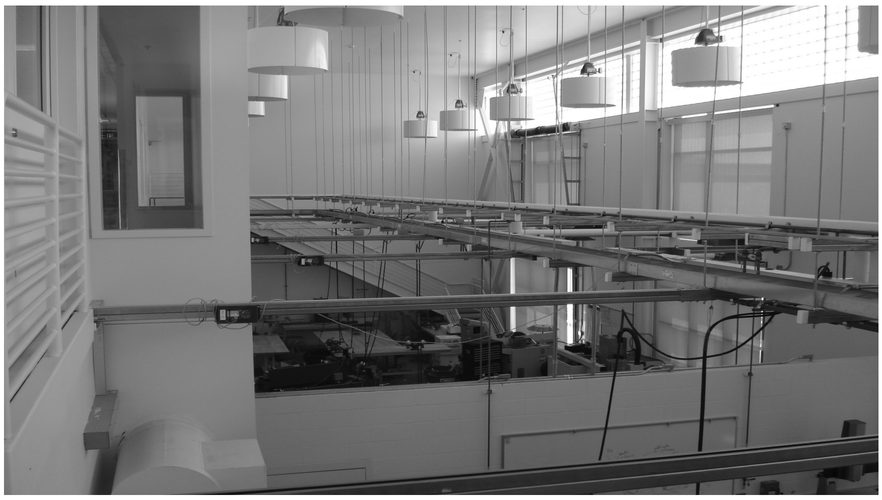


Figure 5.1: SNMS TestBed

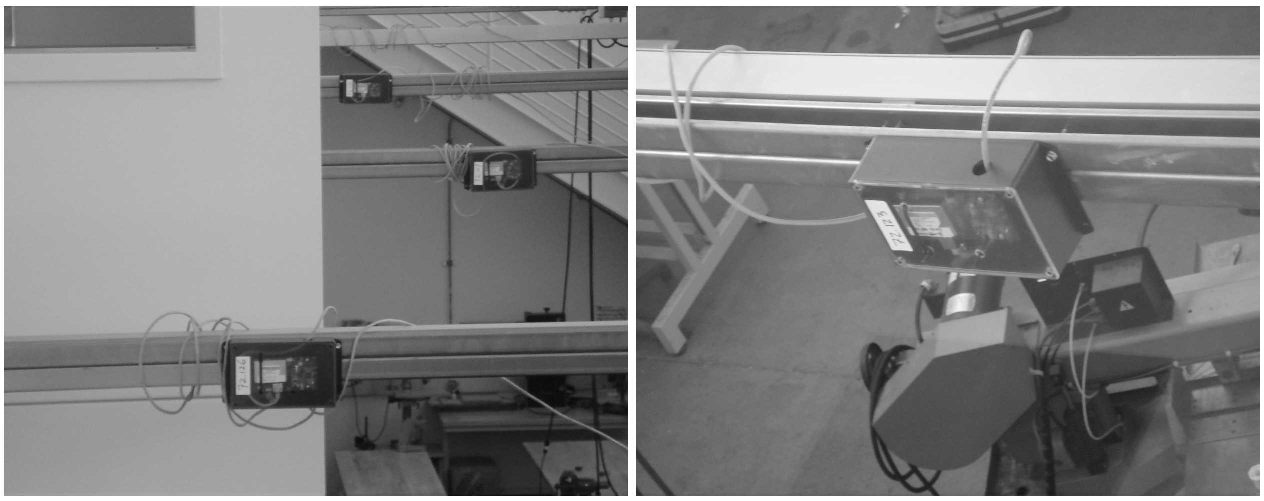


Figure 5.2: SNMS TestBed

In this TestBed, each node composes of two devices: the sensor MICAz and the programming board MIB600, see Figure 5.3.

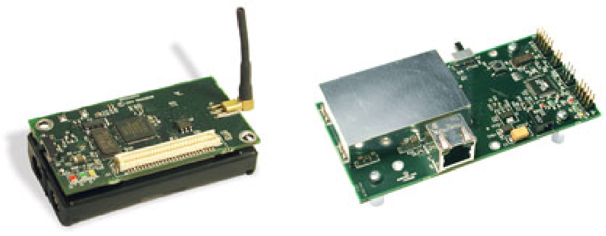


Figure 5.3: (a) MICAz (b) MIB600

The "MICA Mote" family has evolved out of research projects at the University of California at Berkeley, partially with the collaboration of Intel, starting in the late 1990s. Different versions (MICA, MICA2, MICA2Doc, MICAz) have been designed, and are commercially available via the company Crossbow.

The MICAz is a 2.4GHz, IEEE 802.15.4 compliant, Mote module used for enabling low-power, wireless, sensor networks, see Figure 5.3 (a). The MICAz Mote features several new capabilities that enhance the overall functionality of Crossbow's MICA family of wireless sensor networking products. Using TOS, a single processor board (MPR2400CA) can be configured to run your sensor application/processing and the mesh networking radio stack simultaneously. The MICAz IEEE 802.15.4 radio offers both high speed (250 kbps) and hardware security (AES-128). The MICAz 51-pin expansion connector supports Analog Inputs, Digital I/O, I2C, SPI, and UART interfaces. These interfaces make it easy to connect to a wide variety of external peripherals.

A base station allows the aggregation of sensor network data onto a PC or other computer platform. Any MICAz Mote can function as a base station by plugging the MPR2400CA Processor/Radio Board into an MIB510CA serial interface board. The MIB510CA provides an RS-232 serial interface for both programming and data communications.

Crossbow also offers a stand-alone gateway solution, the MIB600CA, for TCP/IP-based Ethernet networks. The MIB600CA provides Ethernet (10/100 Base-T) connectivity to the MICAz/MICA2 family of motes for communication and in-system programming. The MIB600CA allows remote access to sensor network data via TCP/IP. The MIB600CA serial server connects directly to a 10 Base-T LAN like any other network device.

In this TestBed, each sensor node connects to Ethernet through one MIB600CA board for the power supply and for reset in case any serious fault happens. The nodes communicate with each other through RF radio and collect environment status data and send it to sink node through a multi-hop data gathering tree. Any PC that connects to Ethernet and can communicate with the TCP port of the sink node can setup the SNMS client. We start a SerialForwarder on one of the PC in our Sensorweb lab, then multiple SNMS client can be setup on different PCs at the same time.

5.2.2 Software Setup

The implemented SNMS Server is contributed to the directory shown in Figure 5.4: **SNMS** is the server module written in nesC, and **monitor** is the SNMS client which consists of a set of java tools.

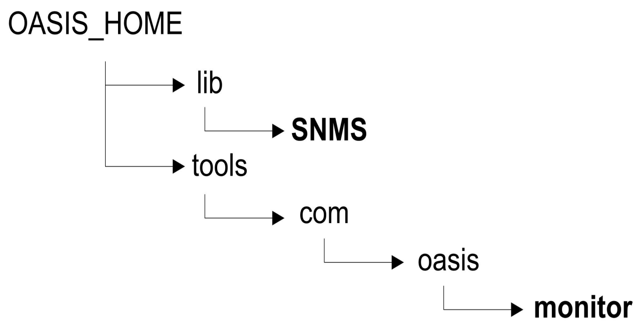


Figure 5.4: Interface of SNMS module

To set up the SNMS server in the node, follow the steps described below:

1. Configure the application to include the SNMS module.

   Before wiring SNMS to the application, first configure the Makefile of the application to include the directory of SNMS related modules, including SNMS, RPC and RamSymbols module.

2. Wire SNMS in the top-level configuration file of the application.

   By wiring with SNMS module, the application has been equipped with the core management mechanisms and can work as a SNMS server. For example, it can call the command sendEvent() in the interface provided by SNSM to report network status to its client. The requirements for the application itself is to provide two communication patterns, collection and dissemination, to SNMS to support remote control. It is because the current version of SNMS depends on the communication module in the application.

3. Customize SNMS for this specific application.

   After wiring SNMS into application, the SNMS server is setup in the node. The other modules in the application have the responsibility to export important network attributes, such as energy consumption level, link quality. It also needs to allow management related functions to be visited remotely by SNMS, such as adjust data rate, turn on/off RFPower. This can be simply done by marking the function with @rpc. Besides the responsibilities, the modules in the application also can take advantage of the management services, such as the WatchDog mechanism, to manage their own process, or customize the contents of the event message to report the fault happened in the application module.

The SNMS client, called the **monitor**, can be easily installed on a PC. There are two options for the installation: compile the source code of the monitor to get an executable file, or directly run the .jar file. The usage of the monitor java tool will be explained in a SNMS demonstration section.

5.3 SNMS Demonstration

As mentioned before, the SNMS client is actually a set of java tools that provide a Graphic User Interface to the network manager to manage the sensor network. The SNMS server will run on the sensor nodes to response to the requests or control commands from SNMS client and act accordingly.

5.3.1 Network Topology and Status Monitor and Control

The panel named "Sensorweb Topology" is the first page of the three-paged MainFrame of monitor tool suite, see Figure 5.5.

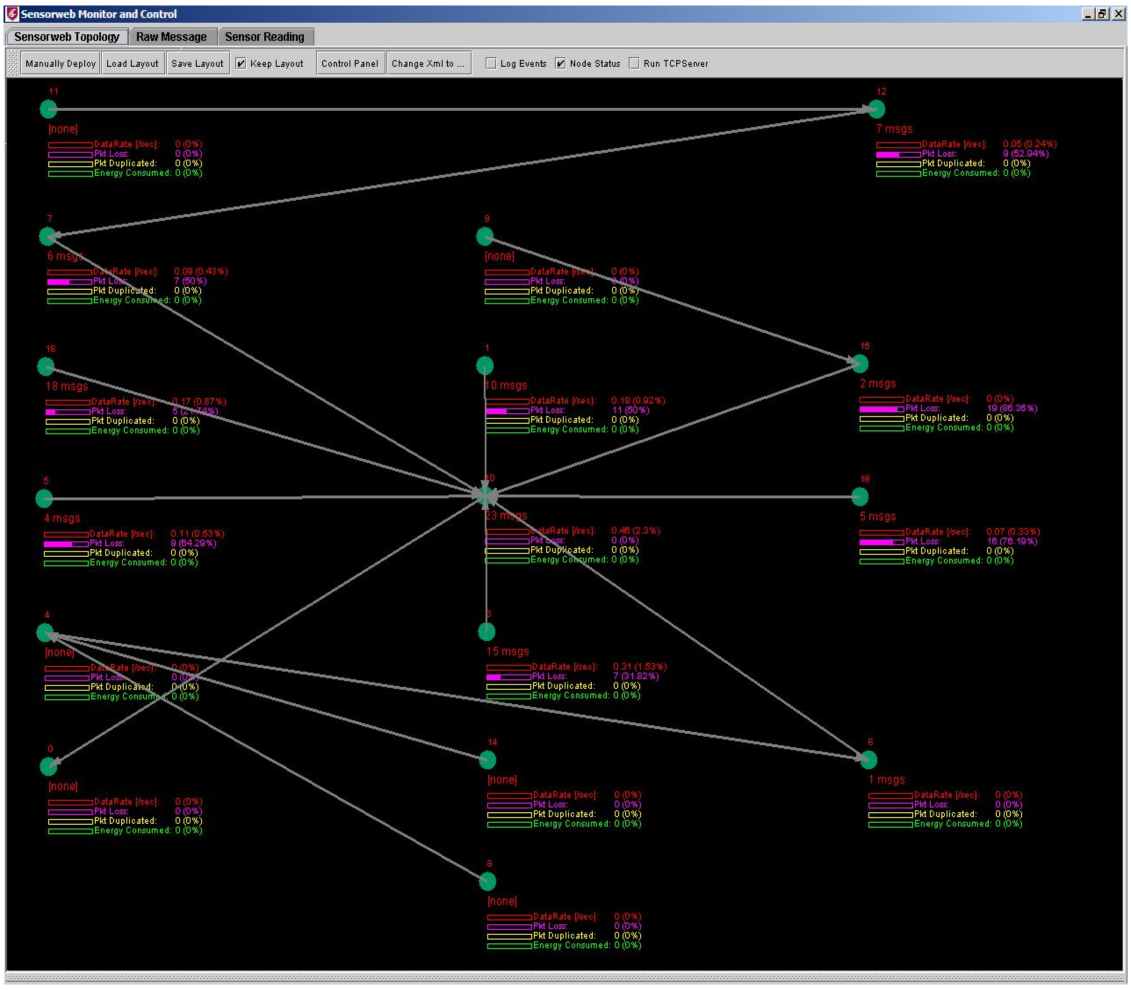


Figure 5.5: Topology monitoring for WSN

The functions provided in this panel is listed as following:

- Monitor real time topology of the whole WSN.

  The topology is formed based on the data packets and routing packets the sink node receives. Whenever a node successfully sends a packet to the sink node, the node (source of the packet) will be displayed on the topology panel, and a link between the sender and the receiver of this packet will also be displayed on the panel. If no packet can be received from a existing node for awhile, the node will be deleted from the panel. In this way, the manager can monitor the connectivity and the real time traffic among the whole network.

The first group of buttons and checkboxs on the ToolBar are used to control the layout of the nodes in the topology. For example, you can switch between "manually deploy" and "fit to screen" to decide the positions of the nodes on the panel; You can also configure the layout using the save/load layout file if the locations of the nodes are fixed (e.g., the TestBed we setup on the ceiling).

- Monitor network status.

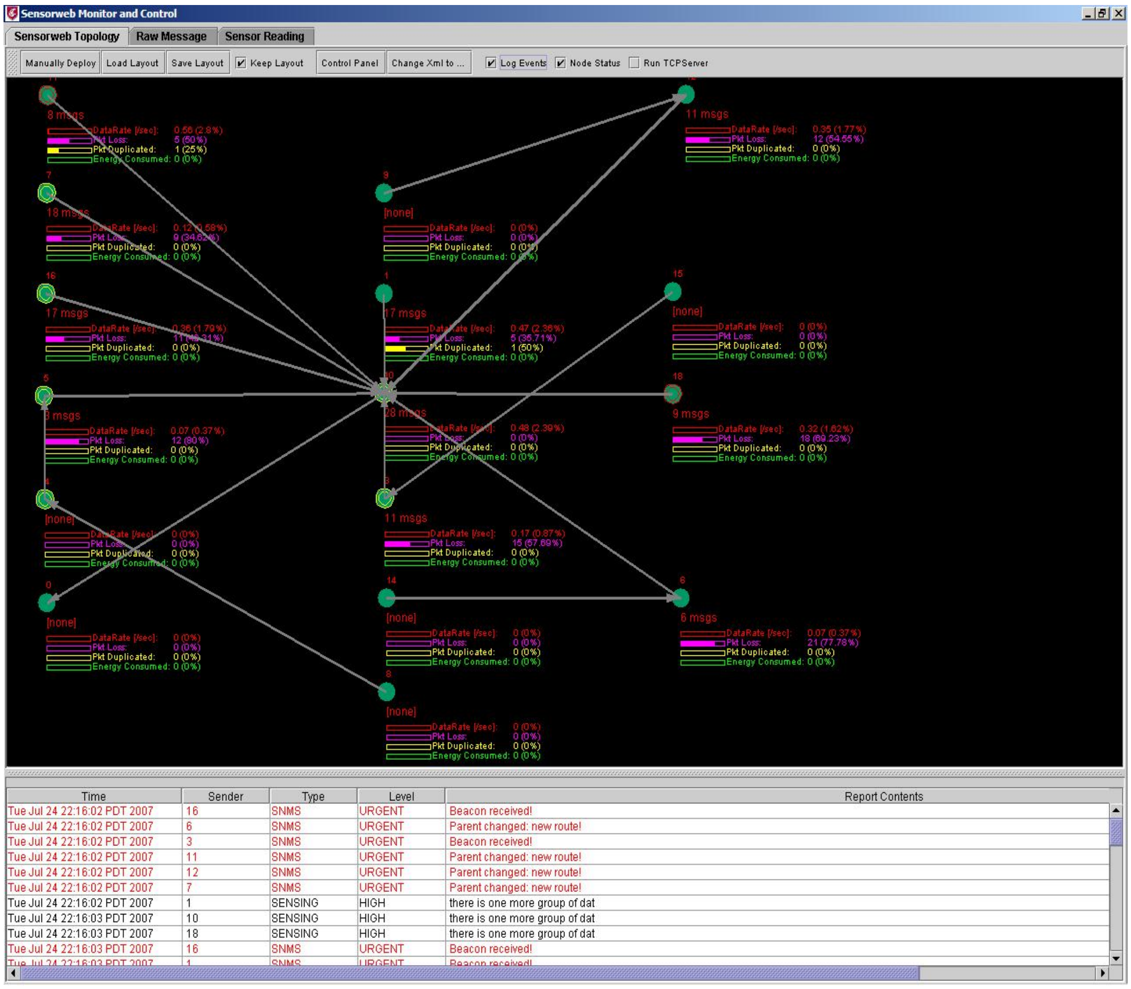
  Beside each node, there is a status box showing the main network status. In the current version of monitor, most of them reflect the quality of the network communication, such as data transmission rate, packet loss rate, and packet duplicated rate. It can be easy to include more sensor node qualities, such as energy consumption, link quality, etc.

  For the properties of monitoring a network with high density, status displaying can be turned off by unchecking the "Node Status" checkbox on the ToolBar.

- Display real time reported events.

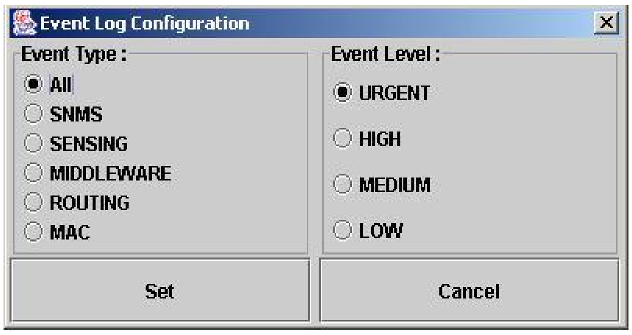
  The "Log Event" checkbox on the ToolBar will control the display of the Event Report panel. The Topology panel with the embedded event report panel is shown in Figure 5.6. In the event report panel, the real time events reported by the modules in the sensor nodes will be listed in the event table.

  When the event report panel is shown in the topology panel, you can click on the table header to set the event report level for different module in the application. The "Event Log Configuration" dialog is shown in Figure 5.7. By default, the red color indicates an urgent



Time	Sender	Type	Level	Report Contents
Tue Jul 24 22:16:02 PDT 2007	16	SNMS	URGENT	Beacon received!
Tue Jul 24 22:16:02 PDT 2007	6	SNMS	URGENT	Parent changed: new route!
Tue Jul 24 22:16:02 PDT 2007	3	SNMS	URGENT	Beacon received!
Tue Jul 24 22:16:02 PDT 2007	11	SNMS	URGENT	Parent changed: new route!
Tue Jul 24 22:16:02 PDT 2007	12	SNMS	URGENT	Parent changed: new route!
Tue Jul 24 22:16:02 PDT 2007	7	SNMS	URGENT	Parent changed: new route!
Tue Jul 24 22:16:02 PDT 2007	1	SENSING	HIGH	there is one more group of dat
Tue Jul 24 22:16:03 PDT 2007	10	SENSING	HIGH	there is one more group of dat
Tue Jul 24 22:16:03 PDT 2007	18	SENSING	HIGH	there is one more group of dat
Tue Jul 24 22:16:03 PDT 2007	16	SNMS	URGENT	Beacon received!
Tue Jul 24 22:16:03 PDT 2007	1	SNMS	URGENT	Beacon received!

Figure 5.6: Event Report Panel



Event Type : [✓ All] SNMS SENSING MIDDLEWARE ROUTING MAC

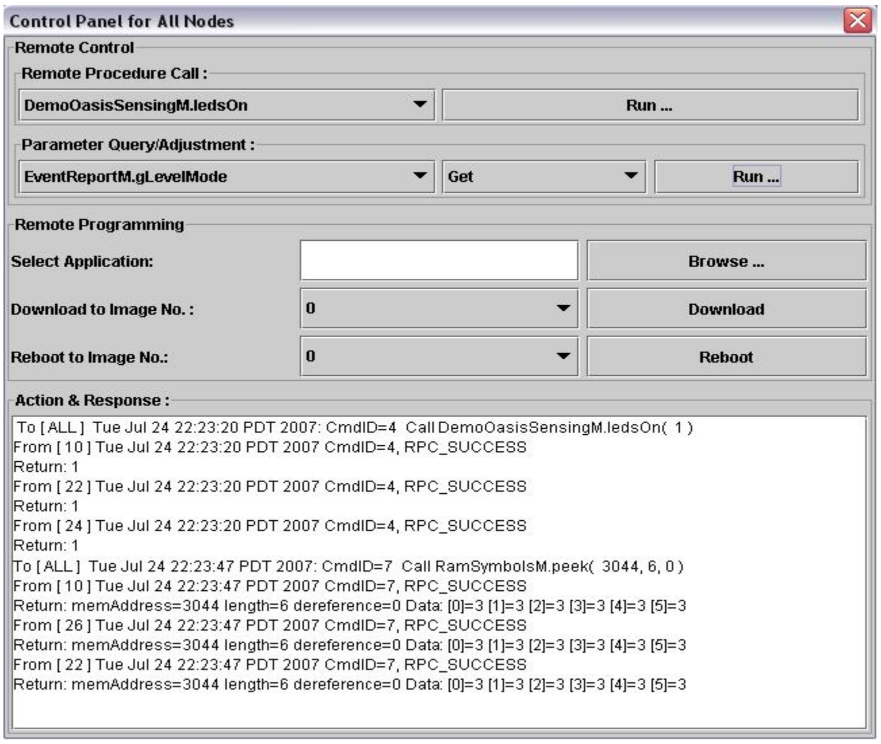
Event Level : [✓ URGENT] HIGH MEDIUM LOW

Figure 5.7: Set Event Report Level

event, which means a severe fault occurred in some module, and will always get through and be reported here. At the same time, all reported event will be logged into event.log file as a monitoring record.

- Remote control sensor nodes.

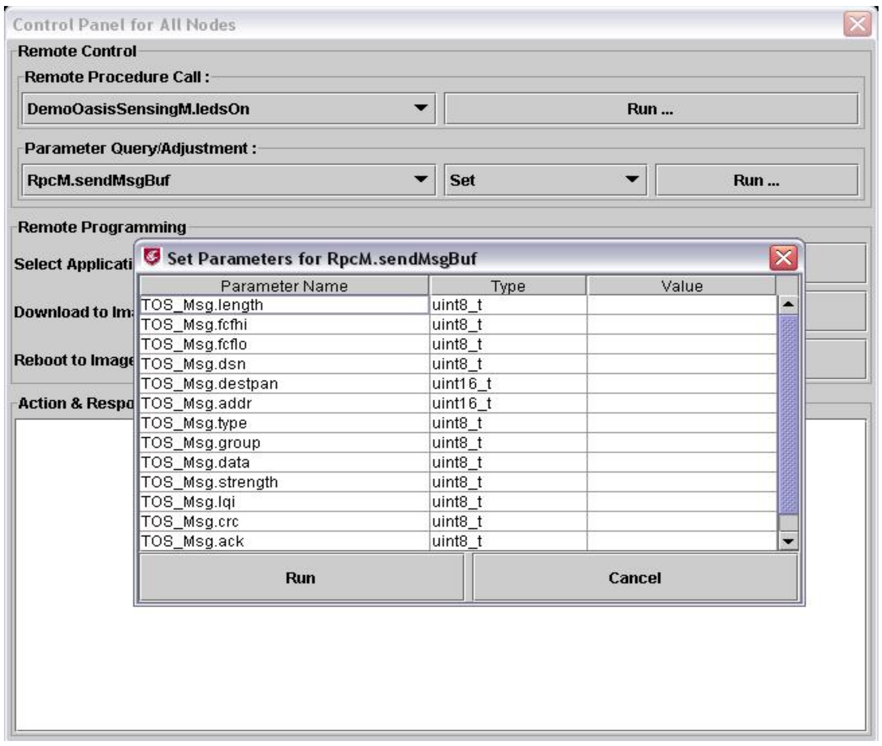
  There are two subfunctions of remote control function in ControlPanel shown in Figure 5.8: Remote Procedure Call, and Parameter Query and Adjustment.



```
 To [ ALL ]  Tue Jul 24 22:23:20 PDT 2007: CmdID=4  Call DemoOasisSensingM.ledsOn( 1 )
From [ 10 ] Tue Jul 24 22:23:20 PDT 2007 CmdID=4, RPC_SUCCESS
Return: 1
From [ 22 ] Tue Jul 24 22:23:20 PDT 2007 CmdID=4, RPC_SUCCESS
Return: 1
From [ 24 ] Tue Jul 24 22:23:20 PDT 2007 CmdID=4, RPC_SUCCESS
Return: 1
To [ ALL ]  Tue Jul 24 22:23:47 PDT 2007: CmdID=7  Call RamSymbolsM.peek(  3044, 6, 0 )
From [ 10 ] Tue Jul 24 22:23:47 PDT 2007 CmdID=7, RPC_SUCCESS
Return: memAddress=3044 length=6 dereference=0 Data: [0]=3 [1]=3 [2]=3 [3]=3 [4]=3 [5]=3
From [ 26 ] Tue Jul 24 22:23:47 PDT 2007 CmdID=7, RPC_SUCCESS
Return: memAddress=3044 length=6 dereference=0 Data: [0]=3 [1]=3 [2]=3 [3]=3 [4]=3 [5]=3
From [ 22 ] Tue Jul 24 22:23:47 PDT 2007 CmdID=7, RPC_SUCCESS
Return: memAddress=3044 length=6 dereference=0 Data: [0]=3 [1]=3 [2]=3 [3]=3 [4]=3 [5]=3
```

Figure 5.8: Remote Control

"Remote Procedure Call" provides a list of all functions running on the sensor nodes that can be called from SNMS client. After clicking the "Run" button, a RPC command will be sent to the node and ask the node to run the selected function, and the result of that function will be returned to the SNMS client and be displayed on the "Action and Response List" panel. Similar for "Parameter Query and Adjustment," instead of a function result, the current value of the selected parameter will be returned and displayed here. The value of parameters on



Remote Procedure Call : DemoOasisSensingM.ledsOn

Parameter Query/Adjustment : RpcM.sendMsgBuf Set

Parameter Name	Type	Value
TOS_Msg.length	uint8_t	
TOS_Msg.fcfhi	uint8_t	
TOS_Msg.fcflo	uint8_t	
TOS_Msg.dsn	uint8_t	
TOS_Msg.destpan	uint16_t	
TOS_Msg.addr	uint16_t	
TOS_Msg.type	uint8_t	
TOS_Msg.group	uint8_t	
TOS_Msg.data	uint8_t	
TOS_Msg.strength	uint8_t	
TOS_Msg.lqi	uint8_t	
TOS_Msg.crc	uint8_t	
TOS_Msg.ack	uint8_t	

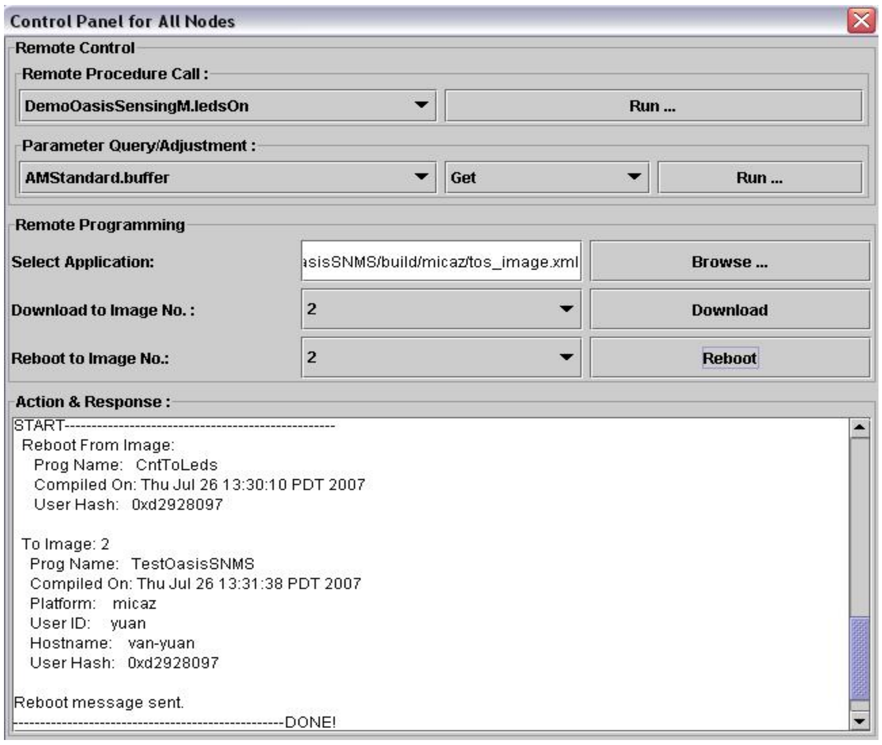
Figure 5.9: Set Parameter by Remote Control

the node can be set from the SNMS client through the ControlPanel too.

As explained in Chapter 3 and 4, The information to support RPC mechanism is abstracted from an XML file, which is generated during compile time of the application. To avoid the faulty RPC action caused by the inconsistency of the XML files used in monitor and for current application running on the nodes, XML version checking is added whenever a RPC command is performed. After you receive “Wrong XML file” error message in the “Action and Response List” panel, you should close ControlPanel, and select the correct XML by clicking “Change XML to ...” button on ToolBar before you run Remote Control function.

- Remote Reprogramming.

  This is another function provided in the ControlPanel. It allows the manager to reconfigure a sensor node by remotely reprogramming it. You can reboot the node using the preloaded programs located in the flash on the node, or download the new application into the node.



Select Application: asisSNMS/build/micaz/tos_image.xml

Download to Image No. : 2

Reboot to Image No.: 2

```
START-------------------------------------------------
 Reboot From Image:
   Prog Name:   CntToLeds
   Compiled On: Thu Jul 26 13:30:10 PDT 2007
   User Hash:   0xd2928097

 To Image: 2
   Prog Name:   TestOasisSNMS
   Compiled On: Thu Jul 26 13:31:38 PDT 2007
   Platform:    micaz
   User ID:     yuan
   Hostname:    van-yuan
   User Hash:   0xd2928097

Reboot message sent.
-------------------------------------------------DONE!
```

Figure 5.10: Remote Programming

5.3.2 *Network Packet Trace and Debug*

This subtool on the second page of MainFrame of monitor traces all types of packets transmitted between the sensor nodes in the WSN,see Figure 5.11.

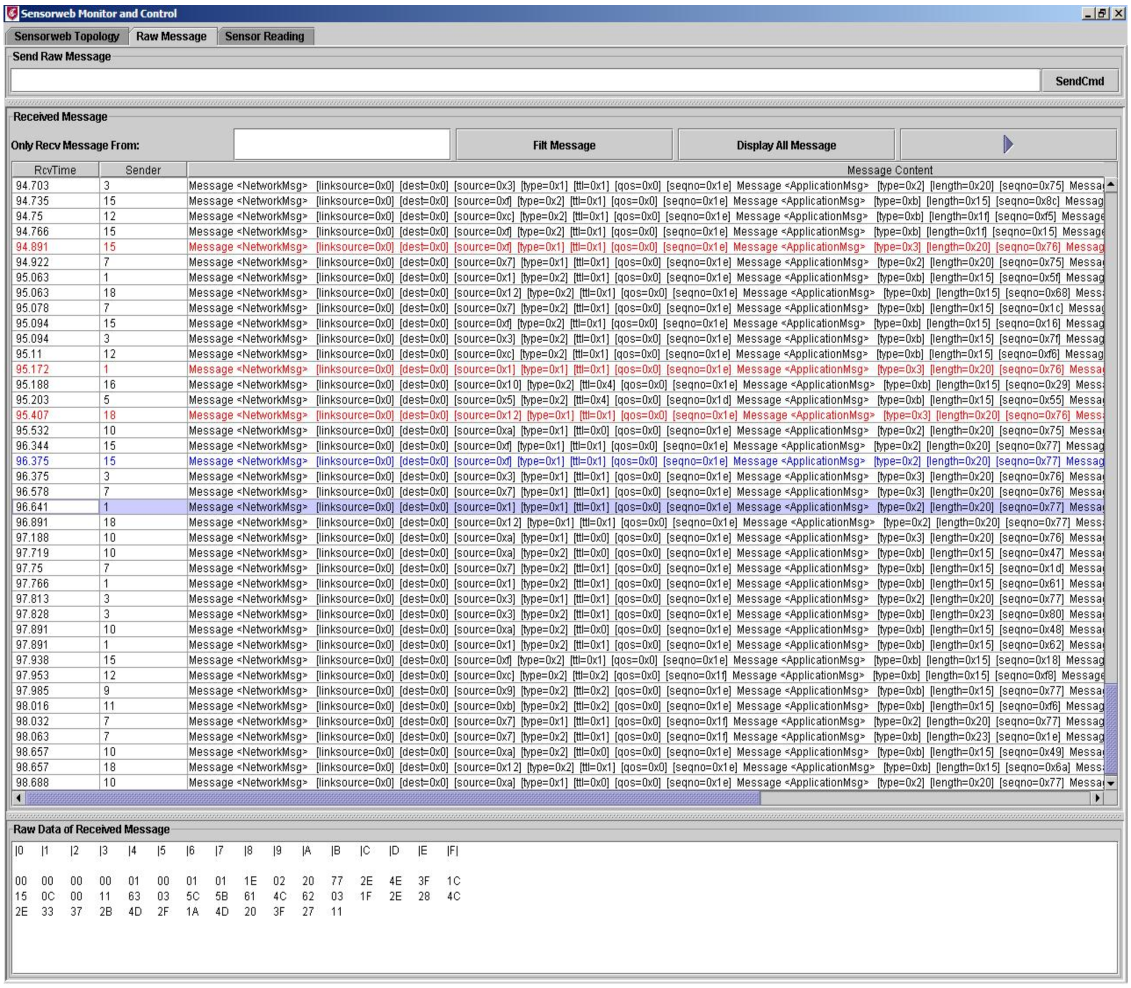
The functions provided in this page includes:

- Display all packets received from all nodes.

  For each packet received in the SNMS client, if the packet type is known, besides the received time and the sender, the content of the packet will be interpreted into packet header and content too; otherwise, no content of the packet will be shown in the packet table.

- Display raw data for selected packet.

  It is useful when the client receives any unknown type of packet. In this case, the user can check the raw data of that packet directly and try to determine the format of it.



RcvTime	Sender	Message Content
94.703	3	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x3] [type=0x1] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0x2] [length=0x20] [seqno=0x75] Messa
94.735	15	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0xf] [type=0x2] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0xb] [length=0x15] [seqno=0x8c] Messag
94.75	12	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0xc] [type=0x2] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0xb] [length=0x1f] [seqno=0xf5] Message
94.766	15	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0xf] [type=0x2] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0xb] [length=0x1f] [seqno=0x15] Message
94.891	15	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0xf] [type=0x1] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0x3] [length=0x20] [seqno=0x76] Messag
94.922	7	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x7] [type=0x1] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0x2] [length=0x20] [seqno=0x75] Messa
95.063	1	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x1] [type=0x2] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0xb] [length=0x15] [seqno=0x5f] Messag
95.063	18	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x12] [type=0x2] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0xb] [length=0x15] [seqno=0x68] Mess
95.078	7	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x7] [type=0x2] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0xb] [length=0x15] [seqno=0x1c] Messa
95.094	15	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0xf] [type=0x2] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0xb] [length=0x15] [seqno=0x16] Messag
95.094	3	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x3] [type=0x2] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0xb] [length=0x15] [seqno=0x7f] Messag
95.11	12	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0xc] [type=0x2] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0xb] [length=0x15] [seqno=0xf6] Messag
95.172	1	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x1] [type=0x1] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0x3] [length=0x20] [seqno=0x76] Messa
95.188	16	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x10] [type=0x2] [ttl=0x4] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0xb] [length=0x15] [seqno=0x29] Mess
95.203	5	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x5] [type=0x2] [ttl=0x4] [qos=0x0] [seqno=0x1d] Message <ApplicationMsg> [type=0xb] [length=0x15] [seqno=0x55] Messa
95.407	18	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x12] [type=0x1] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0x3] [length=0x20] [seqno=0x76] Mess
95.532	10	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0xa] [type=0x1] [ttl=0x0] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0x2] [length=0x20] [seqno=0x75] Messa
96.344	15	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0xf] [type=0x1] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0x2] [length=0x20] [seqno=0x77] Messag
96.375	15	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0xf] [type=0x1] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0x2] [length=0x20] [seqno=0x77] Messag
96.375	3	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x3] [type=0x1] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0x3] [length=0x20] [seqno=0x76] Messa
96.578	7	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x7] [type=0x1] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0x3] [length=0x20] [seqno=0x76] Messa
96.641	1	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x1] [type=0x1] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0x2] [length=0x20] [seqno=0x77] Messa
96.891	18	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x12] [type=0x1] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0x2] [length=0x20] [seqno=0x77] Mess
97.188	10	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0xa] [type=0x1] [ttl=0x0] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0x3] [length=0x20] [seqno=0x76] Messa
97.719	10	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0xa] [type=0x2] [ttl=0x0] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0xb] [length=0x15] [seqno=0x47] Messa
97.75	7	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x7] [type=0x2] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0xb] [length=0x15] [seqno=0x1d] Messa
97.766	1	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x1] [type=0x2] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0xb] [length=0x15] [seqno=0x61] Messa
97.813	3	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x3] [type=0x1] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0x2] [length=0x20] [seqno=0x77] Messa
97.828	3	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x3] [type=0x2] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0xb] [length=0x23] [seqno=0x80] Messa
97.891	10	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0xa] [type=0x2] [ttl=0x0] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0xb] [length=0x15] [seqno=0x48] Messa
97.891	1	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x1] [type=0x2] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0xb] [length=0x15] [seqno=0x62] Messa
97.938	15	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0xf] [type=0x2] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0xb] [length=0x15] [seqno=0x18] Messag
97.953	12	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0xc] [type=0x2] [ttl=0x2] [qos=0x0] [seqno=0x1f] Message <ApplicationMsg> [type=0xb] [length=0x15] [seqno=0xf8] Message
97.985	9	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x9] [type=0x2] [ttl=0x2] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0xb] [length=0x15] [seqno=0x77] Messa
98.016	11	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0xb] [type=0x2] [ttl=0x2] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0xb] [length=0x15] [seqno=0xf6] Messag
98.032	7	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x7] [type=0x1] [ttl=0x1] [qos=0x0] [seqno=0x1f] Message <ApplicationMsg> [type=0x2] [length=0x20] [seqno=0x77] Messag
98.063	7	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x7] [type=0x2] [ttl=0x1] [qos=0x0] [seqno=0x1f] Message <ApplicationMsg> [type=0xb] [length=0x23] [seqno=0x1e] Messag
98.657	10	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0xa] [type=0x2] [ttl=0x0] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0xb] [length=0x15] [seqno=0x49] Messa
98.657	18	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0x12] [type=0x2] [ttl=0x1] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0xb] [length=0x15] [seqno=0x6a] Mess
98.688	10	Message <NetworkMsg> [linksource=0x0] [dest=0x0] [source=0xa] [type=0x1] [ttl=0x0] [qos=0x0] [seqno=0x1e] Message <ApplicationMsg> [type=0x2] [length=0x20] [seqno=0x77] Messa

```
|0   |1   |2   |3   |4   |5   |6   |7   |8   |9   |A   |B   |C   |D   |E   |F|

00   00   00   00   01   00   01   01   1E   02   20   77   2E   4E   3F   1C
15   0C   00   11   63   03   5C   5B   61   4C   62   03   1F   2E   28   4C
2E   33   37   2B   4D   2F   1A   4D   20   3F   27   11
```

Figure 5.11: RawData Page

- Send out Raw Command.

  It is the reverse function of Display raw data. If the user wants to send out any type of packet through the SNMS client by broadcasting, he/she can directly write the raw data of that packet on this panel, and send it out by clicking on “SendCmd” button.

Since this tool traces all the packets, including those other than data packets, such as Routing-Beacon packets, the functions provided here are much more useful for debugging the communication module designed in the application. By checking the packets and their format, the designer can verify the correctness of its mechanisms or strategies.

5.3.3 Real-Time Data Display

It displays the application-related data (i.e., raw data directly abstracted from data packet or derived data defined by user) on a simulated Oscilloscope Screen. It can be used to display the changes happened in the monitored environment, such as volcano environment in the OASIS case, which is the final goal of this environment monitoring sensor network.

Figure 5.12 shows the simulated Oscilloscope with the randomly generated demo data.

5.4 SNMS Testing

After the basic functionalities of the SNMS designed in this thesis have been checked, the validity and efficiency of the following main functions of the SNMS have been tested under different generated scenarios.

- Call Functions on SNMS Server by Remote Control.

  In the scenario of a data gathering application, such as volcano monitoring, there are several sensors collecting different types of data at different rates. The sensing module provides an interface to adjust the sensing rate of each sensor. After exposing this interface to the SNMS client, the user can remotely configure the sensing rate from the GUI of the SNMS client. The changed sensing rate has been verified by checking the data log file. Furthermore, more

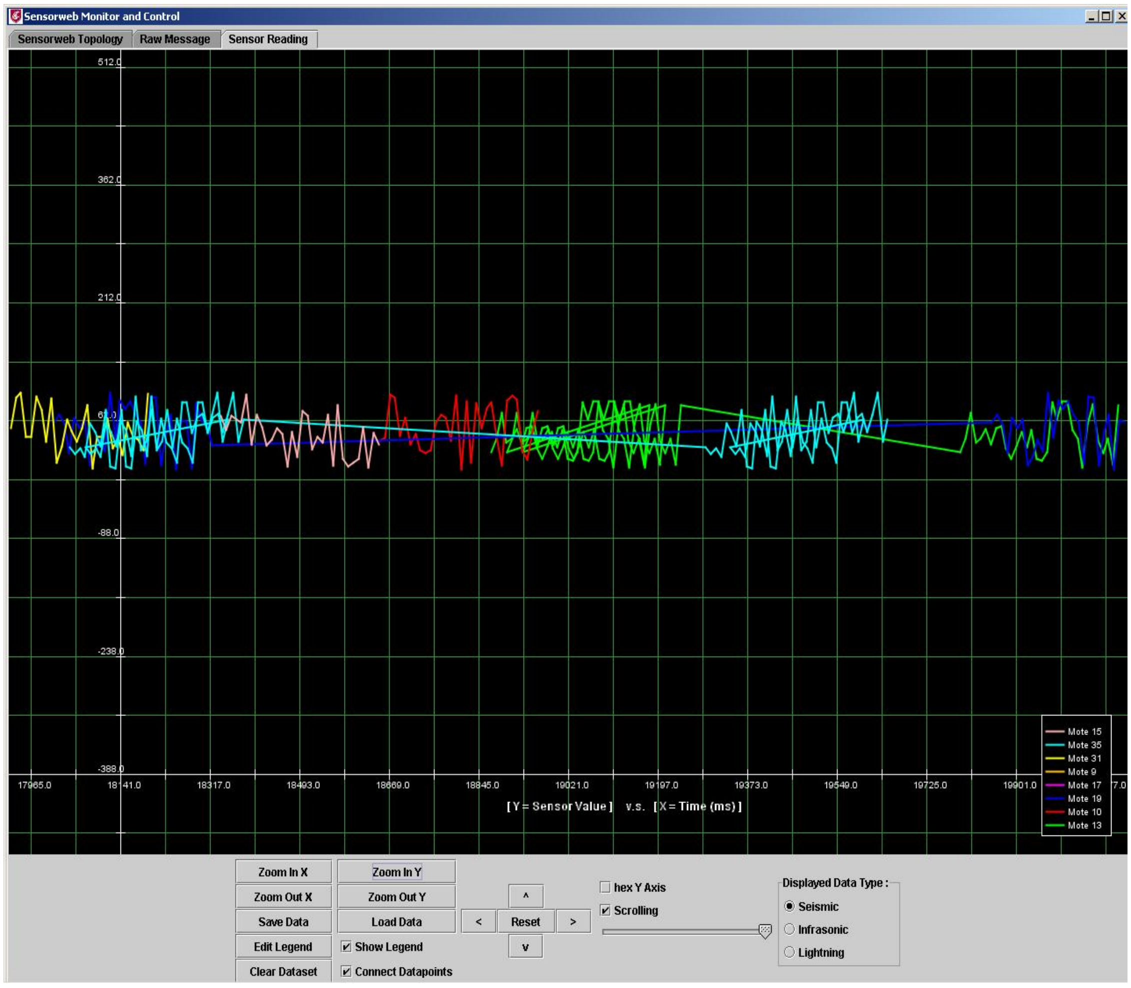


Figure 5.12: Oscope Page

functions with different type of parameters have been used as sample functions to test the remote function calls. The correctness has also been checked.

- Get/Set Parameters on SNMS Server by Remote Control.

  First, several useful multi-hop routing related parameters, such as $currentParent$ of each node, $BeaconPeriod$, and $numSuccessTransmitPacket$ have been retrieved via "Get", the correctness have been checked by comparing with the topology graph and corresponding debug information. Second, the packet sequence number "seqNo" has been changed by "Set", which is reflected by the newly received packets with the new seqNo. Finally, some parameters of different types are sampled to be tested with the Get/Set by remote control, the correctness is also checked accordingly. Note that if the length of the parameter exceeds the maximum length which can be handled by remote control, a warning message will be displayed to notify the user.

- EventReport Function.

  By default, no filter is set for reported events, i.e., all events will be sent to the SNMS client. Several dummy events with different levels and coming from different modules are set to report in the testing application. By using remote configuration for the event report, the user can turn on/off any specified event. The validity of this remote configuration has been checked from the received events listed on the EventReport panel.

- Remote Programming.

  This functionality is tested on a set of MICAz nodes, after they have finished flash formatting and the bootloader has been downloaded onto them (these are required by Remote Programming). Two different applications, called $TestOasisSNMS$ and $CntToLeds$, are remotely programmed into all nodes at the same time through the function provided on the Control-Panel of the SNMS client GUI. After the downloading is finished, the nodes are remotely rebooted to these applications alternatively, and can work correctly after each rebooting.

The problem of current version of Remote Control is, the reliable dissemination routing protocol is under development right now, and the remote control is currently using an unreliable broadcasting protocol. So the remote command cannot be guaranteed to be delivered to all the nodes in the network. In most cases, after a remote command is sent out from the SNMS client, the client cannot get response from all nodes in the network. But it will not be an issue after the implementation of the reliable dissemination routing protocol is done.

CHAPTER SIX

CONCLUSIONS

Due to the continuing advances in network and application design in WSNs, the development of a sensor network management system is becoming necessary and possible. Because the significant differences between traditional networks and WSNs, a different management solution for WSNs is required.

6.1 Main Contributions

In this thesis, a lightweight SNMS based on RPC has been proposed. It considers the unique properties of WSNs, such as limited storage, limited communication bandwidth, and high variability of applications. The main advantages of our SNMS design is:

- It provides a generic framework of SNMS for WSNs, which can be rapidly integrated into applications, and no change for SNMS framework needed when it is added to different applications;
- It allows the user to do the customization for different applications with minimal effort (tagging), and makes the SNMS development for a specific application much simpler;
- The framework is lightweight in terms of meeting the minimal resource requirements on each sensor node, such as storage occupation, bandwidth occupation, etc, and yet still can provide a rich management functionality.

6.2 Future Work

One limitation to the current designed SNMS is the dependency on the communication module of the application. The quality of the Collection and Dissemination communication patterns will directly effect the correctness and efficiency of management functions. This is also a common concern about the effect of unreliable wireless transmission on SNMS. If the routing protocols cannot guarantee against latency or loss, it may lead to unforeseen effects on the application such as unsynchronized behavior between nodes. So more research needed for the supporting reliable dissemination and collection protocols.

Another special issue related to our SNMS design is the security. By providing the RPC function to SNMS, the SNMS client can remotely access a node's functions or variables over a wireless network. The consistency of application information between client and server side is needed to avoid faulty actions. For the current version, a hash-key checking mechanism has been added to SNMS client, which will give a warning message to the SNMS client when the information on both client and server sides are not matched during the remote command is processed. But the consistency has to be maintained by the user himself. Some policy-based security mechanism needs to be added to the current SNMS in the future.

For remote reprogramming, there is no verification mechanism for program authenticity and integrity incorporated in current version of network programming functions in SNMS. Since wireless networks use a broadcast medium, an attacker can easily inject or corrupt a packet which passes the CRCs used by link layers. Such an attack can be detected but not before the entire program has been downloaded. There are already some research working on this problem[13]. We should think about upgrading the reprogramming function in SNMS to a secured one.

Considering the efficiency of RPC calls and responses, current status query is limited by the size of payload of Rpc message, and more flexible query mechanism needs to be developed, e.g., to support multi-page response message, or support compacted message with multiple queries in it to reduce communication overhead.

BIBLIOGRAPHY

[1] Oasis: Optimized autonomous space in-situ sensor web. http://sensorweb.vancouver.wsu.edu.

[2] *Agents for wireless sensor network power management*, 2005.

[3] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38:393–422, March 2002.

[4] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. Developing multi-agent systems with a fipa-compliant agent framework. *Software: Practice and Experience*, 31(2):103–128, 2001.

[5] B. N. Bershad, T. E. Anderson, E. D. Lazowska, and H. M. Levy. Lightweight remote procedure call. *ACM Transactions on Computer Systems*, 8(1):37–55, February 1990.

[6] A. D. Birrell and B. J. Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1):39–59, February 1984.

[7] Athanassios Boulis, Chih-Chieh Han, and Mani B. Srivastava. Design and implementation of a framework for efficient and programmable sensor networks. In *The 1st international conference on Mobile systems, applications and services*, 2003.

[8] Athanassios Boulis and Mani Srivastava. Node-level energy management for sensor networks in the presence of multiple applications: Pervasive computing and communications (guest editors: Mohan kumar, diane cook and anand tripathi). *Wireless Networks*, 10(6):737+.

[9] S. Brown and C. J. Sreenan. Updating software in wireless sensor networks: A survey. Technical report, 2006.

[10] Yao-Chung Chang, S, and Jiann-Liang Chen. Cluster based self-organization management protocols for wireless sensor networks. *IEEE Transactions on Consumer Electronics*, 56(1):75–80, 2006.

[11] W. Chen, N. Jain, and S. Singh. *ANMP: Ad hoc network network management protocol*, 1999.

[12] B. Deb and B.Nath. Wireless sensor networks management, 2005.

[13] Jing Deng, Richard Han, and Shivakant Mishra. Secure code distribution in dynamically programmable wireless sensor networks. 2006.

[14] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the world with wireless sensor networks. In *IEEE ICASSP Conference*, May 2001.

[15] Damianos Gavalas, Dominic Greenwood, Mohammed Ghanbari, and Mike O'Mahony. Advanced network monitoring applications based on mobile/intelligent agent technology. *Computer Communications*, 23(8):720–730, April 2000.

[16] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach to networked embedded systems. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2003.

[17] J. Hill, M. Horton, R. Kling, and L. Krishnamurthy. The platform enabling wireless sensor networks. *Communication of the ACM*, 47(6):41–46, 2004.

[18] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. *SIGOPS Oper. Syst. Rev.*, 34(5):93–104, December 2000.

[19] C. Hsin and M. Liu. A two-phase self-monitoring mechanism for wireless sensor networks. *Computer Communication special issue on Sensor Netowrks*, 29(4):462–476, 2006.

[20] Jonathan Hui. Deluge 2.0 - tinyos network programming, July 2005.

[21] Jonathan Hui and David Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *The 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, 2004.

[22] M. Kahani and H. W. P. Beadle. Decentralized approaches for network management. *Computer Communication Review*, 27:36–47, July 1997.

[23] Holger Karl and Andreas Willig. *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons, Ltd, 2005.

[24] Sukun Kim, Shamim Pakzad, David Culler, James Demmel, Gregory Fenves, Steve Glaser, and Martin Turon. Wireless sensor networks for structural health monitoring. In *The 4th international conference on Embedded networked sensor systems*, 2006.

[25] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Fault-tolerant clustering in ad hoc and sensor networks. In *26th IEEE International Conference on Distributed Computing Systems*, July 2006.

[26] S. S. Kulkarni and M. Arumugam. Infuse: A tdma based data dissemination protocol for sensor networks. Technical report, 2004.

[27] S. S. Kulkarni and L. Wang. Mnp: multihop network reprogramming service for sensor networks. In *The 25th IEEE International Conference on Distributed Computing Systems*, June 2005.

[28] James F. Kurose and Keith Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[29] Ali Lakhia, Hayley Iben, and Rachel Rubin. Watchdog designs for tinyos motes, May 2002.

[30] Winnie L. Lee, Amitava Datta, and Rachel Cardell-Oliver. *Network Management in Wireless Sensor Networks*, chapter 2. 2006.

[31] Winnie L. Lee, Amitava Datta, and Rachel Cardell-Oliver. Winms: Wireless sensor network-management system, an adaptive policy-based management for wireless sensor networks. Technical report, June 2006.

[32] Allan Leinwand and Karen Fang. *Network Management: A Practical Perspective, 2nd Edition*. Addison-Wesley Professional, 2 edition, October 1995.

[33] Zhigang Li, Xingshe Zhou, Shining Li, Gang Liu, and Kejun Du. *Issues of Wireless Sensor Network Management*, pages 355–361. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2005.

[34] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. Wireless sensor networks for habitat monitoring. In *The First ACM International Workshop on Wireless Sensor Networks and Applications*, 2002.

[35] Pedro J. Marrn, Andreas Lachenmann, Daniel Minder, Matthias Gauger, Olga Saukh, and Kurt Rothermel. Management and configuration issues for sensor networks. *International Journal of Network Management*, 15(4):235–253, 2005.

[36] Linnyer B. Ruiz, Jose M. Nogueira, and Antonio A. F. Loureiro. Manna: A management architecture for wireless sensor networks. *IEEE Communications Magazine*, 41(2):116–125, February 2003.

[37] Linnyer B. Ruiz, Isabela G. Siqueira, Leonardo, Hao C. Wong, Jos, and Antonio A. F. Loureiro. Fault management in event-driven wireless sensor networks. In *MSWiM '04: Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 149–156, New York, NY, USA, 2004. ACM Press.

[38] Chien-Chung Shen, Chavalit Srisathapornphat, and Chaiporn Jaikaeo. An adaptive management architecture for ad hoc networks. *IEEE Communications Magazine*, 41(2):108–115, February 2003.

[39] A. Sinha and A. Chandrakasan. Dynamic power management in wireless sensor networks. *IEEE Design and Test of Computers*, 18(2):62–74, 2001.

[40] Siva and B. S. Manoj. *Ad Hoc Wireless Networks Architectures and Protocols*. Communications Engineering and Emerging Technologies. Prentice Hall, 2004.

[41] Hyungjoo Song, Daeyoung Kim, Kangwoo Lee, and Jongwoo Sung. Upnp-based sensor network management architecture. In *The Second International Conference on Mobile Computing and Ubiquitous Networking*, April 2005.

[42] T. Stathopoulos, J. Heidemann, and D. Estrin. A remote code update mechanism for wireless sensor networks. Technical report, Los Angeles, CA, USA, 2003.

[43] Lakshminarayanan Subramanian and Randy H. Katz. An architecture for building self-configurable systems. In *MobiHoc '00: Proceedings of the 1st ACM international symposium on Mobile ad hoc networking & computing*, pages 63–73, Piscataway, NJ, USA, 2000. IEEE Press.

[44] Andrew S. Tanenbaum and Albert S. Woodhull. *Operating Systems Design and Implementation (3rd Edition) (Prentice Hall Software Series)*. Prentice Hall, January 2006.

[45] Gilman Tolle and David Culler. Design of an application-cooperative management system for wireless sensor networks. *2nd European Workshop on Wireless Sensor Networks*, January 2005.

[46] Kamin Whitehouse, Gilman Tolle, Jay Taneja, Cory Sharp, Sukun Kim, Jaein Jeong, Jonathan Hui, Prabal Dutta, and David Culler. Marionette: Using rpc for interactive development and

debugging of wireless embedded networks. In *IPSN2006: The Fifth International Conference on Information Processing in Sensor Networks*, April 2006.

[47] Z. Ying and X. Debao. Mobile agent-based policy management for wireless sensor networks. In *IEEE WCNM Conference 2005*, September 2005.

[48] Mengjie Yu, Hala Mokhtar, and Madjid Merabti. A survey of network management architecture in wireless sensor network. In *The 6th Annual PostGraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting*, June 2006.

[49] J. Zhang, E. C. Kulasekere, K. Premaratne, and P. H. Bauer. Resource management of task oriented distributed sensor networks. In *The 2001 IEEE International Symposium on Circuits and Systems*, May 2001.