Sixth Annual IEEE International Conference on Pervasive Computing and Communications

A Lightweight Sensor Network Management System Design

Fenghua Yuan* Wen-Zhan Song* Nina Peterson† Yang Peng* Lei Wang* Behrooz Shirazi†
Richard LaHusen‡

***Abstract*— In this paper, we propose a lightweight and transparent management framework for TinyOS sensor networks, called L-SNMS, which minimizes the overhead of management functions, including memory usage overhead, network traffic overhead, and integration overhead. We accomplish this by making L-SNMS virtually transparent to other applications hence requiring minimal integration. The proposed L-SNMS framework has been successfully tested on various sensor node platforms, including TelosB, MICAz and IMote2.**

***Index Terms*— wireless sensor network, network management, lightweight, transparent, remote procedure call, event report**

I. Introduction

Recent advances in hardware, sensor, and wireless network technologies are enabling large-scale deployment of continuous wireless data acquisition systems, called Wireless Sensor Networks (WSNs)[2], at a fraction of the cost of the previous cost. In typical sensor network applications, the sensor nodes monitor fields and generate events which travel hop-by-hop toward one or more sink nodes allowing users to interact with the sensor network through commands. Thus, the user is able to control the network through commands sent from the sink node(s) to the rest of the network. WSNs are utilized in many different scenarios, including battlefield surveillance, environment monitoring and medical applications.

WSNs play an increasingly important role in supporting a wide range of applications, however, along with this, many design and implementation challenges must be addressed. WSNs usually operate in hostile environments, hence the individual nodes are typically difficult (if not impossible) to configure in the field. Thus, network maintenance, reconfiguration, and recovery [19] mechanisms must be developed to enable effective wireless network management. WSNs should also be able to provide an autonomous real time network topology of the deployed nodes, which is critical in disaster scenarios. In addition, WSN must be able to reveal to its users whether or not a deployed network is functioning properly. A WSN should also be able to record all of the node failures and events in real time[17]. The design of the network management system for WSNs is much more difficult than for traditional wired networks [14], [13], due to the limited memory and computing resources of the nodes, limited bandwidth, unreliable links, and diverse application requirements.

Motivated by these needs, we propose a light-weight and transparent sensor network management framework. We have successfully tested the proposed mechanism using our OASIS project [1]. The goal of OASIS is to develop a prototype dynamic and scalable hazard monitoring *sensor web*, which will be deployed on Mount St. Helens - an active volcano in Southern Washington, U.S.A. In the OASIS project, various geophysical and geochemical sensors will generate continuous high-fidelity data which will be delivered to the control center at WSU Vancouver and further linked to both USGS and NASA JPL. One key feature of this WSN is that it is a smart sensor network capable of adjusting its resource utilization based on both the volcanic situation as well as the external inputs, including satellite monitoring (EO-1) and scientific analysis.

The rest of the paper is organized as follows. In Section II, we review the related works of current sensor network management systems. In Section III, we present the two key components of L-SNMS: a transparent command and control mechanism and a transparent and configurable event report mechanism. In Section IV, we describe the implementation of L-SNMS in TinyOS sensor networks. The results of the functionality and validity tests of our L-SNMS are presented and evaluated in Section V. Finally, the conclusions are presented in Section VI.

II. Related Work

Simple Network Management Protocol (SNMP) [12], [11] developed in the late 1980's by ISO, is the standard management technique for traditional TCP/IP networks. However, there are several unique characteristics of WSNs that make SNMP unapplicable to WSNs. First, the communication overhead associated with SNMP is too great for WSNs. Second, SNMP requires that each sensor node maintain a large MIB (Management Information Base) which is impractical for storage-constrained sensor nodes. Finally, sensor-specific failures, which are common in WSNs, are not handled by SNMP.

The *Ad Hoc Network Management Protocol* (ANMP)[5] and Guerilla[16] are two protocols designed for managing mobile wireless ad-hoc networks, however they can only be implemented in some specific types of WSNs. ANMP uses the hierarchical clustering of nodes in order to reduce the number of messages exchanged between the manager and the agents. ANMP is an extended SNMP which includes MIB extensions, dynamic configuration of agents, dynamic extension of the agents, and

This research is supported in part by NASA ESTO AIST Grant NNX06AE42G.

*School of Engineering and Computer Science, Washington State University, Vancouver, WA.

†School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA.

‡Cascades Volcano Observatory, U.S. Geological Survey, Vancouver, WA.

0-7695-3113-X/08 $25.00 © 2008 IEEE
DOI 10.1109/PERCOM.2008.36

an application-specific security module. The main contribution of ANMP is that it enables SNMP to 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.

The *Management Architecture for Wireless Sensor Networks* (MANNA) [15], is a management solution specifically for WSNs, which adopts ad hoc network management techniques. MANNA provides a general framework for policy-based management of sensor networks by collecting dynamic management information from the MIB, which it then maps into a WSN model. The WSN model is executed though management functions and services. However, one of these functions, MIB update a centralized operation, is extremely expensive in terms of the large amount of energy it consumes.

The *Sensor Network Management Protocol*, sNMP [6] is a management framework proposed by Deb *et al.*, which defines sensor models that represent the current state of the network and defines various network management functions. Additionally, sNMP provides algorithms and tools for retrieving network state information through the execution of the network management functions; however, it a MIB-based framework and like MANNA it suffers from similar drawbacks.

The *Sensor Network Management System* (SNMS), proposed by Tolle and Culler [17], 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 querying system allows users to collect and monitor physical parameters of the nodes environment. The event-driven logging system allows the user to set event parameters which allow nodes to report their data only if they have meet the specified event thresholds set by the user. L-SNMS improves the SNMS [17] by incorporating an RPC (Remote Procedure Call) mechanism inspired by an interactive debugging tool for WSN development, Marionette [18]. Our RPC mechanism additionally provides the user with the necessary support needed in order to access the functions and variables of applications running on the sensor nodes during runtime. Thus, the design of a lightweight generic SNMS framework is possible.

III. L-SNMS Design

Wireless sensor network design is usually application dependent, since different applications may have different requirements. However, it is ideal to provide generic network management component support, making it customizable and extensible to different applications. The most commonly required application management functions which are included in our L-SNMS architecture are performance monitor and control, configuration management, fault management, and data management. All of these management modules are designed to be application-independent, and provide generic SNMS services. For example, the performance management module supports remote inquiry into all of the network performance related parameters in any application. Therefore, when applied to different applications, the network manager can simply select the specific parameters which are of particular importance for the specific application and customize the L-SNMS accordingly. In the following, we will present the two key mechanism in our L-SNMS to support such a generic and transparent framework.

A. Lightweight Remote Procedure Call

A Remote Procedure Call (RPC)[4], [3], [18] allows the PC to access the functions and variables of a 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 nodes. Through these objects, the node's functions can be called and its variables can be read and written. With RPC, embedded applications seamlessly span from the PC to the sensor nodes.

Figure 1 shows the generation process of an RPC during compile time, and the Remote Procedure Calls during run time. The compile time actions are supported by TinyOS 1.x [18], which add the RPC function stub to the SNMS server. The run time actions are implemented by L-SNMS allowing the formation of a run time generic MIB for remote control.

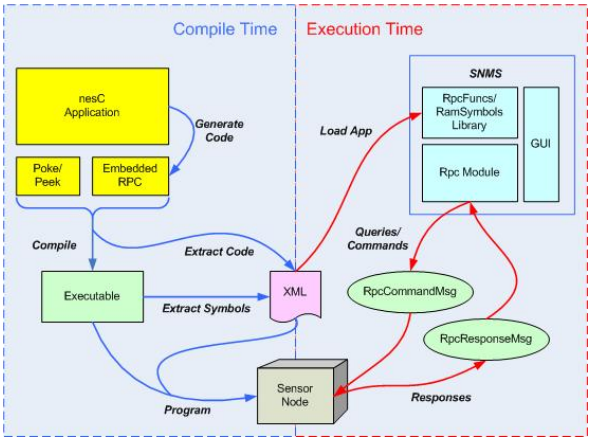

Fig. 1. The RPC solution to reduce the overhead of sensor node management.

1) Compile Time Actions: At compile time, an RPC server stub is automatically generated and added to the nesC application. The hooks for each RPC function which are marked with an "@rpc" tag are automatically added to the RPC module wired to the application. This mechanism makes our SNMS design transparent to all other applications; thus, user only needs to add an "@rpc" tag after the manageable interfaces (which makes it callable from the control center). For instance, to export the data sensing configuration functions for remote control, all that a user needs to do is appending an "@rpc" to the interface SensingConfig.

```
interface SensingConfig @rpc();
```

Then, all functions in the interface definition of SensingConfig can be called by the remote program.

```
interface SensingConfig {
  command result_t setStatus(uint8_t sensorType,
        bool trigger);
  command uint8_t getStatus();
  command result_t setRate(uint8_t sensorType,
```

```
        uint16_t samplingRate);
  command uint8_t getRate(uint8_t sensorType);
}
```

Simultaneously, all of the RPC client's necessary information is use to parse the application source code and exported it to an XML file by a Perl program. The extracted nesC declarations, including the data structures, typedefs, message formats, module names, and interface names are written to the XML file called *nescDecls.xml*.

2) Execution Time Actions: At run time, the RPC client imports all of information from the XML file. Next, the user sends commands to and receives responses from the nodes through the RPC client over a multi-hop routing layer.

Since all of the data is transmitted over the network using the native style RPC server, it is mandatory that the RPC client convert the data to the CPU architecture format on the PC. In addition to the traditional types, the RPC client also supports complex types such as arrays and structures. The RPC client can also automatically parse nested message structures, such as the addition of header bytes to a packet by the MAC and routing protocols. The work done by the RPC client relieves a significant burden from the RPC server, relinquishing its previous serialization and deserialization responsibilities.

Since the RPC client is capable of generating variables and messages of the same types as those used on the sensor nodes, it is thus capable of reading the RPC interface definition and making new client objects with functions identical to those on the nodes. These functions take nesC style variables as arguments and, when called, pack and unpack the arguments and response messages, respectively, which provide a basis for the interactions between the user and the sensor nodes.

While all of the variables of all of the nesC modules are available (by default), due to the cost of importing all of functions, only the functions or interfaces marked with an "@rpc" tag will be imported. Thus, in order to import a function the user is only required to tag the function.

In the traditional network management design, in order for a centralized SNMS to be able to perform network management, both SNMS client and the server must maintain a local database MIB, which contains information about the properties and the services that the SNMS server supports including the current configuration, operation statistics, and parameters to be controlled by the managed resources. The SNMS server also implements packet types and Get/Set functions on its MIB variables. The MIB located at the SNMS client contains network management information extracted from the MIBs of the managed entities (sensor nodes) in the network. It is the responsibility of the SNMS server to store and update its MIBs, receive control packets, and perform the necessary computations for specific management actions.

With our L-SNMS, there is no need for a MIB in SNMS servers (sensor nodes), because all the management information is abstracted at compile time, and stored in the SNMS client (the powerful PC). In order to minimize the strain on the sensors, only a small portion of code (which will call the corresponding functions or peek/poke variables) is stored in each node. Additionally, L-SNMS also minimizes the computational overhead of the MIB management on the sensor nodes. When an inquiry is made from the SNMS client, the sensor nodes respond by getting the value of the inquired parameter directly from the client based on the memory address provided in the inquiry message. This process eliminates the need for unnecessary storage of parameters on the nodes as well as the need for unnecessary computations by the nodes, keeping L-SNMS lightweight.

B. Run-time Configurable Event Report Mechanism

It is often necessary for the sensor network to report interesting events. For example, a collection tree component periodically sends specific debug messages including the current parent, the current link estimates, and the current path costs. This data is then used to build a graph of the network in real-time and study its stability. Usually, these debug messages are only enabled at compile time, and once enabled, they are constantly sent at a fixed rate. This design is not flexible or desirable should be able to be manipulated by the developer as needed.

Our event report mechanism is controlled by flags (switches) and capable of changing the reporting frequencies of the RPC. There are two types of event messages supported by our report system. The first is a one-time urgent event, which is used to report any faults happening on the sensor nodes, while the second is a periodic event message. The report rate of the periodic event message is configurable during run time and can even be turned off when the traffic in the network is high. Additionally, this design also provides the user with the flexibility to extract the system status with different levels of detail simultaneously alleviating the overhead of the application activities. The definitions of the `EventReport` and the `EventConfig` interfaces are:

```
interface EventReport {
  command uint8_t eventSend( uint8_t etype, uint8_t elevel,
     uint8_t *content);
  event result_t eventSendDone(TOS_MsgPtr pMsg,
     result_t success);
}
interface EventConfig {
  command result_t setReportLevel(uint8_t type, uint8_t level);
  command uint8_t getReportLevel(uint8_t type);
}
```

To use these interfaces in TinyOS, the user simply needs to add `uses interface EventReport` to the module, wire it to the SNMS module, and then simply call function `EventReport.eventSend` to report any event to a gateway. This is similar to the way the `trace` function is used.

```
call EventReport.eventSend(EVENT_TYPE_SNMS, EVENT_LEVEL_URGENT,
    eventprintf(``parent:\%i",gbCurrentParent));
```

Our SNMS module exports the `EventConfig` interfaces as RPC, hence the user can choose to report more or less events during run time.

In this section, we propose a generic framework which only requires minimal effort to customize the generic SNMS for a specific application. First, all variables on the head of all nesC modules are accessible by default in L-SNMS without any execution time overhead, and the user can freely select any relevant parameters to perform different management schemes. Second, the user only needs to mark the commands on the SNMS server which he/she wants to exposed to the SNMS client. Third, the

event report system in L-SNMS supports the formatted string (similar to the format definition in printf() in C language), leaving the flexibility of the definition and interpretation of the reported events to the user. L-SNMS requires no additional semantics to be imposed on the generic framework.

IV. Implementation

To implement the SNMS design proposed in Section III, two parts need to be developed separately: the SNMS server and the SNMS client. The SNMS client was developed in a high-level OOP language, Java, and is run on a powerful PC. The implementation of the SNMS server is more complicated due to the critical embedded operating system in which it is developed, and the special properties of the developing language it uses.

The operating system TinyOS [9], along with the programming language nesC [7], form a powerful and relatively easy to use platform in which we implemented both the core operating system functionalities as well as communication protocol stacks and application functions. Experience has shown [7] that in fact programmers do use these paradigms and arrive at relatively small, highly specialized components that are then combined as needed, proving modularity. Also, the code size and memory requirements for this system are quite small. The applications of WSNs are usually run on a collection of wireless sensor nodes together with a base station (PC). For our design the base station serves as the control center, with the SNMS client running on it. The sensor nodes are the SNMS servers.

The hardware which can be utilized is flexible since there are currently many different sensor nodes available for use in wireless sensor network research and development [8], such as the MICA and TMote family motes. When choosing hardware components for a wireless sensor, the application's requirements play a key role in determining the size, cost, and energy consumption requirements of the nodes.

A. Sensor Node Software Component

In our L-SNMS design, the SNMS server has the responsibility of providing the information requested by its clients, and performing the appropriate actions as specified by the commands sent by its clients. The server also has the self-managing functionality which is used to trigger one-time urgent reports when faults occur. In addition to these specific services and functions, the implementation of the SNMS server must also be generic, flexible and expendable. We took advantage of the interface design supported by the nesC language to design our L-SNMS. Figure 2 shows the top-level configuration of an application which is wired to the L-SNMS.

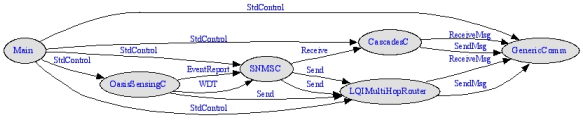

Fig. 2. Apply SNMS to OasisSensing Application

This figure was automatically generated by a TinyOS graphics tool based on the configuration file TestOasisSNMS.nc. Each circle in the figure represents a top-level component labeled with its name. The solid arrow represents a connection (wiring) between two components. For example, the $OasisSensing$ component is connected to the $SNMSC$ component by two arrows, this means that $OasisSensing$ uses the two interfaces, WDT and $EventReport$, provided by the $SNMSC$ module. The $SNMSC$ component provides support for the network management services to other modules. The routing module $LQIMultiHopRouter$ and the $CascadesC$ module provide collection and dissemination routing for the $SNMSC$ module. The $GenericComm$ component is the link layer communication module. The components of the $SNMSC$ module are shown in Figure 3.

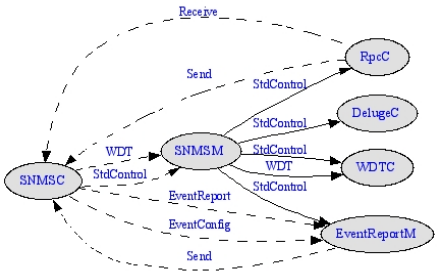

Fig. 3. Component Architecture of SNMS Server

The following submodules are included in the SNMS server to manage specific services: EventReportM, DelugeC, WDTC, and RpcC. The doted arrow signifies that the interface is not directly provided by the connected component. For example, the $RpcC$ component uses two interfaces, $Send$ and $Receive$ from the $SNMSC$ component. These two interfaces are not implemented in the $SNMSC$ component, instead the $SNMSC$ component is used as an intermediate component in order to connect the $RpcC$ component to the real provider of these two interfaces.

B. PC Client Software Component

The SNMS client running on the PC provides the users with the management services of the sensor networks. A Base Station (Sink or Root) sensor node is required to provide a bridge between the PC (client) and the other sensor nodes (servers). All of the packets received by the Base Station from the nodes are forwarded to the PC through the serial port. The Base Station can also forward the packets to its TCP port. The client is capable of being set up from any PC which is connected to the Ethernet. A key piece of the SNMS client implementation is the setting up of communication between the PC and the sensor network, since the server and the client, which are written in different languages (one in nesC, and the other in Java), and are running on different platforms (one on an embedded system of sensor nodes, and the other in cygwin on a PC). MoteIF of TinyOS 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 another means of connectivity.

In L-SNMS, the SNMS client is a set of java tools organized in a multi-tabbed GUI framework. A moteIF object is created at the same time that the GUI is created, and the MessageListeners are distributed amongst these tools. Each tool is triggered by the packets received through these MessageListeners and acts accordingly, refreshing the displayed network topology graph, updating the network status, and logging the packets into the record file.

The panel named "Sensorweb Topology" is the first page of the three-paged MainFrame of the monitor tool suite, Figure 4, monitoring the real time topology and status of the WSN, and displaying the real-time events. The events can be configured during run time through the "Event Log Configuration" dialog shown in Figure 5. "Remote Procedure Call" and "Parameter Query and Adjustment" can be accessed through the Control-Panel.

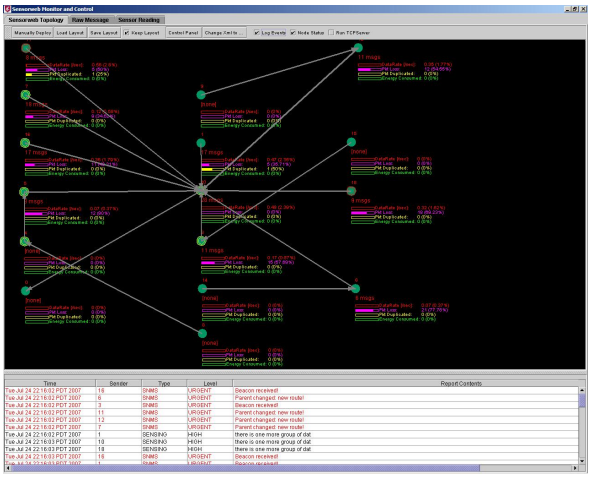

Fig. 4. Event Report Panel.

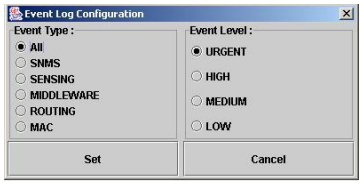

Fig. 5. Set Event Report Level.

Another function provided in the Control Panel allows the manager to reconfigure a sensor node by remotely reprogramming it. This allows the user to reboot a node to the preloaded programs, or download a new application image to the node. This network reprogramming is supported through TinyOS 1.x's Deluge [10].

The subtool on the second page of MainFrame of monitor traces all of the packets transmitted between the sensor nodes in the WSN (raw data). Since this tool traces all of the packets, including RoutingBeacon packets and unknown packets types, the functions provides useful debugging information.

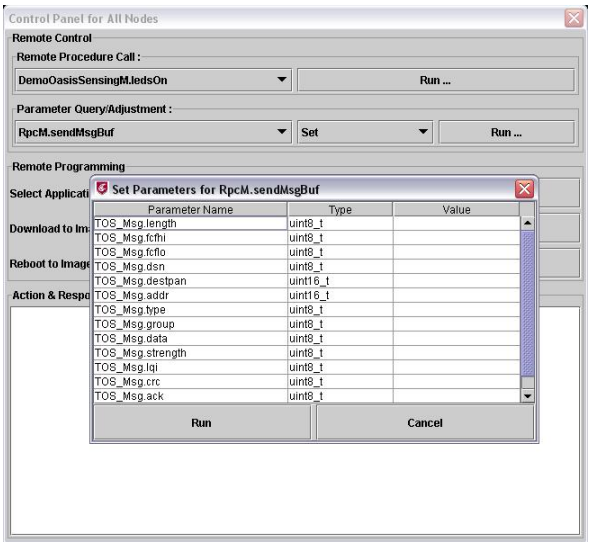

Fig. 6. Setting Parameters by Remote Control.

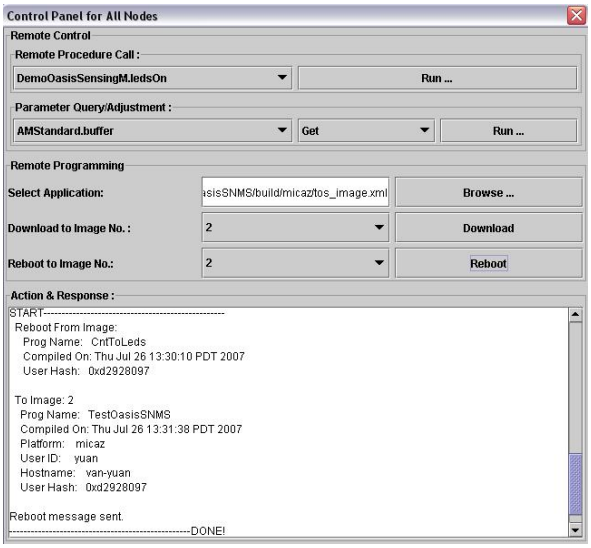

Fig. 7. Remote Programming.

V. Test and Evaluation

As previously mentioned, we tested L-SNMS using our OASIS system, which is equipped with many different sensors which report seismic, infrasonic, and lightning data to the sink at different frequencies. There are two main reasons for adding L-SNMS to our OASIS development, first, it provides us with a way for the network manager to be able to monitor the network status, and make sure that it functioning properly. Second,L-SNMS allows the geologists to adjust the focus of the sensor network to certain types of data or specific spots of the volcano which are of particular interest. An in-door TestBed was setup for the software testing of our OASIS project. It consists of a 17-node wireless sensor network installed on the ceiling of the VELS building on the WSU Vancouver campus. In this TestBed, each node is composes of two devices: the MICAz

sensor and the MIB600 programming board.

After the basic functionalities of the L-SNMS were tested, the validity and efficiency of the following main functions of the L-SNMS were tested separately under different scenarios.

Remote Function Calls: In data gathering applications there are several sensors collecting different types of data at different rates. The sensing module provides an interface to adjust the sensing rates of each sensor. After exposing this interface to the SNMS client, the user can remotely configure the sensing rates from the GUI of the SNMS client. The changed sensing rates have been verified by checking the data log file. Furthermore, functions with different types of parameters were used in order to test the remote function calls.

Remote Get/Set Parameters: First, several useful multihop routing related parameters, such as $currentParent$, $BeaconPeriod$, and $numSuccessTransmitPacket$ were tested via "Get", and the results compared with the topology graph and corresponding debug information. Second, the packet sequence number "seqNo" was changed through the "Set" parameter, which is reflected by the newly received packets with the new seqNo. Finally, parameters of different types were sampled and tested with the Get/Set parameters through the remote control.

EventReport Function: By default, no filters are set for reported events so that all of the events are sent to the SNMS client. Several dummy events with different levels, coming from different modules, were set to be reported in the test applications. Using the remote configuration, the user was able turn on and off specified events. The validity of this remote configuration was confirmed using the received events listed on the EventReport panel.

Remote Programming: The functionality of the remote programming component was tested on MICAz nodes, which had their flash formatted and the bootloader downloaded. Two different applications, $TestOasisSNMS$ and $CntToLeds$, were remotely programmed onto all of the nodes simultaneously through the function provided on the ControlPanel of the SNMS client GUI. After the downloading was finished, the nodes could remotely rebooted.

Based on the RPC mechanism, the size of L-SNMS module on each sensor node is 558 bytes (MICAz motes have 4k of available RAM). The total size of the data gathering application (basic version of the OASIS application) that the L-SNMS is managing is 3257 bytes. As an important functioning module in the application, our L-SNMS only occupies a small portion (around 1/6) of total application size. When compared to traditional management mechanisms, L-SNMS saves approximately 1K of RAM because it does not store the performance related parameters on the sensor nodes. In order to support the RPC mechanism, only 160 bytes of RAM (out of 558 bytes) are needed for buffering packets. Additionally, 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 sensor nodes, the code size of our L-SNMS on the sensor node is only 2924 bytes of ROM out of 19744 bytes (the total size of the testing application).

VI. Conclusion

This paper proposes a lightweight SNMS (L-SNMS) based on an RPC mechanism. Our L-SNMS takes into account the unique properties of WSNs, including limited storage, limited communication bandwidth, and application uniqueness. The main advantages of our L-SNMS design are its generic framework, ease of integration, and lightweight framework. Functions of L-SNMS, such as remote function calls, remote Get/Set, event report, and remote reprogramming, have been extensively tested in a real testbed consisting of 17 MICAz nodes. We have also shown that our L-SNMS significantly reduces the overhead when compared with traditional SNMS systems. We believe that L-SNMS is extensible to many more applications as the increase in the deployment of WSNs continues.

References

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

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

[3] 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.

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

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

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

[7] 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.

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

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.