

Aus dem Institut für Telematik
der Universität zu Lübeck

Direktor
Prof. Dr. Stefan Fischer

Management Support and CPU Scheduling Performance Enhancement in Wireless Sensor Networks

Inauguraldissertation
zur
Erlangung der Doktorwürde
der Universität zu Lübeck

-Aus der Technisch-Naturwissenschaftlichen Fakultät-

Vorgelegt von
Fadi Tirkawi
aus Hama, Syrien

Lübeck, im 2009-03-25

Zum Druck genehmigt
Mündliche Prüfung am 03.08.2009

Prof. Dr. Jürgen Prestin

Management Support and CPU Scheduling Performance Enhancement in Wireless Sensor Networks

Submitted to the University of Lübeck, in partial fulfillment of the requirements for the degree of Doktor-Ingenieurs (Dr. -Ing)

Abstract

This thesis is focused on issues relating to the management of the operating systems in wireless sensor networks. It deals with the questions like: Why the management components are required in operating systems for sensor nodes? What services could be managed in the operating system for such platforms? What are the management constraints of sensor nodes?

TinyOS is one of the mostly used operating systems in this area. TinyOS features multithreading architecture, a very flexible networking stack, and virtual machine implementations. In this thesis, we demonstrate a new scheduling algorithm for TinyOS which especially takes routing issues into account. The algorithm increases the sensitivity and responsiveness in wireless sensor networks. We have modified the priority scheduling algorithm which solves the well-known problem of overloading in simple FIFO scheduling. This scheme improves the fairness of the distributed sensing in the network particularly when the network is subject to mobility, to higher sensor network density and to changes in the topology.

In this thesis, we also present the design and the implementation of a new management tool for wireless sensor network. This tool provides management system framework for the operating system components on sensor nodes. It also provides performance management by providing easy means to observe parameters whenever it is required to test dynamic changes on the nodes. The tool also analyzes the obtained results in human-understandable and user-friendly way which is highly demanded for WSNs. Moreover, configuration management support makes our tool suitable for providing methods to interact with the system running on the sensor nodes.

Content

Content.....	vii
List of Figures.....	xiii
Chapter 1 Introduction.....	1
Chapter 2 Wireless Sensor Network and their Operating Systems	7
2.1. Design Methodologies of WSN Operating Systems.....	8
2.1.1. Control Loop.....	8
2.1.2. Event-based OS.....	9
2.1.3. Legacy OS.....	9
2.1.4. TinyOS-type of OS	9
2.2. Design Considerations of Operating Systems for WSNs	9
2.2.1. Programming Languages	10
2.2.2. System Resources Management	10
2.2.3. Real-Time Requirement.....	11
2.2.4. Application Specificity	11
2.2.5. Algorithms Independency.....	11
2.2.6. Process, Threads and Scheduling.....	11
2.3. Sensor Node Hardware	12
2.3.1. MICA Series	12
2.3.2. BTNodes Series	13
2.3.3. ESB Series	14
2.3.4. Sun Sensor Nodes	14
2.3.5. MarathonNet Sensor Nodes (PaceMate).....	15
2.3.6. iSense Sensor Nodes.....	15
2.4. nesC.....	17
2.4.1. Characteristics of nesC	17
2.4.2. Code Optimization in Wireless Sensor Nodes.....	19
2.5. TinyOS (TinyOS multithreading Operating system).....	20
2.5.1. TinyOS Structure	21

2.5.2. Components in TinyOS.....	22
2.5.3. The inner Structure of a Component.....	24
2.6. Energy Consumption in Wireless Sensor Networks.....	25
2.7. Networking the Nodes in Wireless Sensor Network	27
2.7.1. Networking Stack Characteristics.....	27
2.7.2. Active Message Model (AM) in TinyOS.....	28
2.7.3. Efficient MAC Layers for Sensor Nodes.....	30
Chapter 3 CPU-Scheduling in WSNs' Operating Systems	32
3.1. Scheduling Model in TinyOS	32
3.2. Priority Scheduling in TinyOS.....	33
3.3. The Deadline Scheduler	35
3.4. Topology-Sensitive Priority Scheduling.....	37
3.4.1. Results.....	39
Chapter 4 Wireless Sensor Networks Management	46
4.1. Definition of Network Management.....	46
4.1.1. Configuration Management	47
4.1.2. Fault Management	47
4.1.3. Performance Management	47
4.1.4. Accounting Management	47
4.1.5. Security Management	47
4.2. WSN Management Challenges.....	48
4.3. Centralized vs. De-centralized Management in WSNs.....	49
4.3.1. Centralized Management in WSN:	50
4.3.2. De-Centralized Management in WSN:	50
4.3.3. Hyper-Centralized Management in WSN:.....	51
4.4. Management Models in Wireless Sensor Networks	51
Chapter 5 Challenges and Related Work.....	53
5.1. Motivations and Challenges.....	53
5.1.1. The limited accessibility of WSNs	54
5.1.2. Frequent Node Failures.....	54
5.1.3. Weak Computation and small available Memory.....	55

5.1.4. General WSNs Interfaces.....	55
5.2. Related Work	57
5.2.1. MANNA	57
5.2.2. Nucleus (SNMS):.....	58
5.2.3. Surge Network Viewer:	59
5.2.4. Listen Tool:.....	60
5.2.5. Oscilloscope:.....	60
5.2.6. Trawler:.....	61
5.2.7. Xnp & Deluge:.....	62
5.2.8. BTnode Tools.....	63
Chapter 6 Requirements, Architecture and Features of ITSN.....	66
6.1. Design Requirement.....	66
6.1.1. Information Dissemination	66
6.1.2. Information Collection.....	68
6.1.3. On-Client Dispatching Interface Layer.....	69
6.2. Architecture of the designed Tool.....	70
6.2.1. PC-Side Components	70
6.2.2. Nodes-Side Components.....	71
6.2.3. Communication between Tools Parts (on PC & Sensor Nodes).....	73
6.3. The Main Features of the Designed Tool	76
6.3.1. Generic Management Applications in WSNs.....	76
6.3.2. The Generic Management Nature in the Management Framework.....	79
6.3.3. Management Efficiency of the Framework	80
6.3.4. Fault Tolerance Model of the Framework	81
6.3.5. Plug-ins	85
Chapter 7 Implementation	88
7.1. Main Classes of the PC-Side Components	88
7.2. Plug-ins Implementation.....	94
7.2.1. Plug-in Interface.....	94
7.3. Main Classes of the Nodes-Side Components	97
Chapter 8 Evaluation of the Management Framework.....	102

8.1. Experiment Setup.....	102
8.1.1. Samples Aggregation.....	103
8.1.2. CPU Scheduling Management and Performance Analyses Using the Management Tool.....	107
8.1.3. Map Services and Localization.....	111
8.1.4. Network Management.....	113
8.2. WEB-Based WSN Viewer Application.....	116
Chapter 9 Conclusion.....	119
9.1. A New CPU Scheduling Scheme.....	119
9.2. New Management Framework.....	119
9.3. Efficiency.....	120
9.4. Generality.....	120
9.5. Architecture and Implementation.....	120
9.6. Plug-ins.....	120
9.7. Real World Evaluation.....	121
9.8. Future Potential Work.....	121
References.....	123
Appendix A.....	133
Appendix B.....	135

List of Figures

Figure 2.1	Representation of MICA wireless sensor node platform.....	13
Figure 2.2	BTNodes in ETH Zurich.....	13
Figure 2.3	ECR, ESB nodes at the FU Berlin, CST group.....	14
Figure 2.4	Sun nodes.....	15
Figure 2.5	Pacemate sensor node.....	15
Figure 2.6	iSence sensor node.....	16
Figure 2.7	Compilation operations from nesC to binary file.....	17
Figure 2.8	TinyOS structure composed of a scheduler and a graph of components..	22
Figure 2.9	Component architecture.....	24
Figure 2.10	AM active message model (left), traditional communication model (right)	29
Figure 2.11	Periodic listen and sleep.....	31
Figure 3.1	The CPU-Scheduling priority model in TinyOS.....	33
Figure 3.3	Tinyviz simulator.....	40
Figure 3.4	Results conducted from TOSSIM simulator for a sensor node that increasingly forwards samples from its sub nodes.	41
Figure 3.5	CPU-Scheduler queues of a node forwarding packets for 7 sub-nodes, before and after applying the CPU-scheduling scheme.....	43
Figure 3.6	Standard deviation of packets received from nodes on the base station...	44
Figure 3.7	Affect of changing the percentage of switching according to number of sub nodes.....	45
Figure 4.1	MANNA Architecture.....	48
Figure 5.1	Nucleus structure.....	59
Figure 5.2	Oscilloscope.....	61
Figure 5.3	Trawler.....	62
Figure 5.4	SNIF, Sensor network inspection framework.....	65
Figure 6.1	A representation of a collection tree.....	68
Figure 6.2	General architecture of the management Tool.....	71

Figure 6.3	Architecture of the sensor node including the management components.	72
Figure 6.4	Communication modes in the management tool.....	73
Figure 6.5	Communication between the tool's parts	75
Figure 6.6	Message generation for the tool using MIG	80
Figure 6.7	Sensing maps of temperature and illumination of a forest deployment...	83
Figure 6.8	Node topology of three nodes are performing the fault tolerance scheme	84
Figure 6.9	CPU utilization of a node.....	87
Figure 7.1	UML diagram of the main classes inside the management tool	89
Figure 7.2	UML diagram of the visualization classes inside the management tool...	91
Figure 7.3	UML diagram of events classes inside the tool	93
Figure 7.4	UML diagram of the plug-ins	95
Figure 7.5	Messages structure of MAC control and data messages.....	98
Figure 7.6	Network management agent messages structure	99
Figure 7.7	Scheduling agent messages structure.....	101
Figure 7.8	Messages structure of the management agent for stopping and functioning the nodes	101
Figure 8.1	XML file of the stored sensing parameters.....	104
Figure 8.2	charts of samples of a sensor node generated by the tool.....	105
Figure 8.3	Sampling rate dialog box	106
Figure 8.4	XML file of CPU utilization data stored by the tool	107
Figure 8.5	Topology of the network for demonstrating the CPU-scheduler management feature.....	108
Figure 8.6	CPU-Scheduler performance while reducing the number of the neighboring nodes.....	110
Figure 8.7	CPU-scheduler performance in two cases: A-Left. Performance while functioning and while the node is disabled B-Right. Disabling sub-nodes	110
Figure 8.8	Snapshot of the Map plug-in.....	111
Figure 8.9,	fetching the map of potential WSN deployment	112
Figure 8.10	the routing tables taken from a screen shot from the NetworkPlugin.....	114
Figure 8.11	An XML file shows the nodes and their parent nodes.....	114

Figure 8.12	Snapshot of RSSI and LQI of a node while increasing the distance to the base station.....	115
Figure 8.13	Sensor network viewer.....	116
Figure 8.14	Sensor network viewer left frame.....	117
Figure 8.15	Chart produced by the sensor network viewer.....	118
Figure 9.1	CPU Scheduler Simulator.....	136
Figure 9.2	Execution flow of the queues.....	137

Chapter 1

Introduction

Humans have always tried to increase and extend their awareness capabilities by using different remote sensing techniques. Moreover, the need of controlling and automating human life tasks are driving towards reducing down the sizes of sensing, actuating, computation and communication units so that these units can be integrated into the near or the remote environment. Decreasing the power consumption of these units is also required with that. In addition, the new life style requires that these units should be connected through suitable communication methods that do not restrict their mobility. These units need to be ubiquitous and pervasive which means, it is possible to use and exploit them every where. Such units should be able to interoperate seamlessly, invisibly and autonomously.

Initially, wired sensor networks were introduced to accomplish the above described objective. A wired sensor network is a network of sensor nodes connected through wires. Each node has a processor, few sensors and small memory. These nodes collaborate to solve assigned tasks. Moreover, they serve as a gateway to the data of the physical world, in order to provide users with information about the environment characteristics. Wired sensor networks have been used in a wide range of applications such as industrial monitoring, medical care, home automation, operating surveillance etc. However, wired sensor networks are only realistic for the applications with a small number of nodes. In many applications, such as sustainable bridges, gold or coal mining, barrier surveillance and habitat monitoring, wiring a large number of nodes is infeasible. Therefore, a new area called wireless sensor networks, or in short WSNs, has emerged to replace the wired sensor networks to provide better coverage with higher resolution and many other advantages.

The recent advances in electronics, sensorics and telecommunication have resulted in a variety of sensors types, several processors and different communication techniques. Therefore, there is a big variance of WSN hardware platforms with different capabilities. Moreover, there is a wide range of potential settings of wireless sensing applications in different fields. Therefore, it is difficult to characterize WSNs with precise properties and characteristics. However, we still can specify the challenges which exist in the major part of WSNs applications. These challenges are described in the following points.

- **Small Physical Size, Low Power Consumption and Low Costs:**

The small size of nodes is the first feature in wireless sensor networks. Sometimes, the size of a node is the deciding factor for using a particular type of node in the application. The single node size varies generally from a small box to a small particle.

Regarding the power consumption, an earlier definition of wireless sensor network stated that sensor nodes are supposed to function for months or even years without re-charging their batteries. The nodes stay in sleeping mode most of the time and wake up after a certain interval of time, which could be minutes or hours, to take samples and return back to the sleeping mode. On the contrary, in wireless sensor network for medical [3] or industrial applications, the nodes survive with their energy budget only for few days and at most for weeks due to the high power consumption, which is caused by the high sampling rate.

The low cost of nodes is another key factor because the sensor nodes, in some applications, have to be disposable. They are deployed once without recollecting them again. Normally, the cost of a single node varies from a few cents, according to the future expectations, to hundreds of euros depending on the simplicity of the nodes and the scale of the network.

- **Hardware Limitations**

Each node in the wireless sensor network has very limited resources. Memory size can be, sometimes, as little as just a few kilobytes depending on the application. The communication transmission is very limited in range and power. Furthermore, computing is very weak because the processor in the node, normally, is only a few MIPS capable. These limitations often limit the use of sophisticated algorithms or schemes.

- **High Variety in the Design and Usage**

In future support wireless sensor network technology promises solutions in many areas such as health monitoring, home intelligence, industry, commercial applications, disaster relief, military surveillance etc. A brief overview of the taxonomy of wireless sensor network applications is provided in the Appendix A. These different applications are based on many processing capabilities, communication methods, sensing methods and multiple energy supplying means. This high variety makes general solutions for sensor network applications difficult task.

- **Fault Tolerance:**

Robustness in design is a very important issue since nodes in wireless sensor networks often operate in rough dynamic environments such as catastrophic areas, toxic zones and disasters. Therefore, robust construction must be ensured especially in some applications in which some nodes do not survive. For this reason many, features such as self-maintenance [5] or self-protection characteristics have been proposed for WSN applications.

Wireless sensor networks, being networks, are similar to MANETs. MANET-abbreviation of mobile add-hoc networks-is a type of wireless networks in which the nodes are mobile and dynamically connected without a prior configuration or equipped infrastructure. Some of the similarities between WSNs and MANETs are the ad-hoc deployment, multi hop communication nature, the wireless links and the scarce resources. However, sensor networks still differentiate from MANETs in many aspects. Differences are characterized in the following points [5]:

- *Number of nodes:* MANETs normally have just a small number of nodes, while sensor networks may extend to hundreds or even thousand of nodes.
- *Node density:* density is low in MANETs.
- *Broadcast communication:* it is peer to peer communication in MANETs, but it is mostly broadcast communication in WSNs because of the large node density.
- *Power supply:* in sensor networks, non-rechargeable batteries are used in most of the applications, while in MANETs either direct power source or rechargeable batteries are in use.
- *Data rate:* in general, it is very low in sensor networks, while in MANETs high data rate is in use.

- *Mobility*: it is relatively low in most of WSNs applications, but it can be very high in MANETs.
- *Addressing*: often, in WSNs, there is no unique ID for each node and most of the times, the data-centric naming schemes are used. In MANETs, globally unique ID is required.

In wireless sensor networks, although much excellent research has been done to study the electronic and networking parts of the nodes, there are still many open questions for the researchers. Some of these are for example, finding suitable and generic operating system components, QoS, efficient energy conservation paradigms, topology control, mobility management and others.

In the initial phase of our research work, the focus was on the operating system components of the sensor nodes in WSNs, especially CPU scheduling and routing issues. We have proposed a new CPU-scheduling scheme [1], [2], and [8]. This scheme is an improvement of the priority scheduling algorithm [7], which is an approach known for solving the overloading problem of the simple FIFO CPU-scheduling in a sensor node. The priority scheduling algorithm [7] ensures higher packets throughput, but does not guarantee that the nodes closer to the base station will get fair chance of sending their own packets when the networking operations will be prioritized over the local operations in sensor nodes. This is because of the medium congestion around the base station. Moreover, these nodes not only have to perform the packets forwarding operations but also, in addition, to process their own local packets. The concept of priority switching in the priority scheduling approach is introduced, so that such nodes can perform local tasks as well.

In the course of the verification process of our proposed CPU-scheduling algorithm, many management applications were required in order to configure some parameters of the nodes on the fly and to get debugging results. However, the existing management applications were inefficient and insufficient. Some of these tools which we have used are Surge [9], Oscilloscope [10], Deluge [11] and Xnp [10], Listen [10], Trawler [12] and Nucleus [14].

WSN Management tools should provide control and management of the system components such as the network layer, operating system parameters and application

settings in real sensor nodes. However, such tools that can provide adequate remote interactions of the nodes are really missing. This lack of interaction and management capabilities of such tools is the reason which has motivated us to focus on finding management solutions for WSNs. Thus, the second and the main objective of this thesis is to introduce a new management framework which mainly increases the interactivity with sensor nodes, and provides monitoring of the network as a whole as well as for every individual node. Moreover, this framework aims at fulfilling and accomplishing the issues, to have more setting choices (management) and more elaborative graphical interfaces (visualization). The framework is referred to as management tool or as management application, in this document.

The dissertation is organized in nine chapters. The second chapter introduces operating systems for wireless sensor networks. As the designed tool is integrated with these operating system services, these services are briefly discussed as well. A brief description about the mechanisms, parameters and policies, which can be managed and controlled by the designed tool, are also dealt in this chapter. In addition, some of the existing WSN hardware platforms are listed.

The third chapter describes the existing CPU scheduling schemes used in WSNs. It also discusses our proposal “routing-sensitive priority CPU-scheduling in sensor nodes”.

As our proposed tool is used as a management tool, the fourth chapter describes the management areas in traditional networks and WSNs. Chapter five explains why management tools are required for WSNs. Furthermore, it characterizes the existing tools, their properties and capabilities. In chapter six, the structure of the designed tool is discussed in detail. Communication techniques and some other issues related to the memory usage are also mentioned in this chapter. Moreover, it describes the main features supported by the management tool. Especially, it is focused on the generality and the efficiency of the tool. The Plug-in feature supported by our tool and the designed Plug-ins are mentioned here.

Chapter seven describes the tool implementation, the services that are offered by the tool, and the way of designing additional Plug-ins.

Chapter eight contains evaluation of the designed tool by presenting a real life scenario of the tool. Finally, the last chapter concludes the dissertation. A summary of the dissertation and the potential extension of our work are mentioned here.

Chapter 2

Wireless Sensor Network and their Operating Systems

Operating systems for wireless sensor networks are illustrated in this chapter in order to provide, in the following chapters, a clear picture of how our tool controls the operating system and what the managed parameters could be. Moreover, in order to have a good understanding, in further discussions, of how the management tool is integrated with the operating system. Furthermore, different sensor node hardware platforms are here discussed. Information about algorithms and services running on the sensor node are also provided here.

Normally, embedded devices are operated by software programs called operating systems. These software programs control and manage the resources. In general, there are two categories of operating systems for embedded devices as follows:

- Firmwares are for custom embedded devices such as digital cameras or home appliances controllers. In such devices, few of the operating system services are merged with the application code. The application code, including these services, runs directly on the hardware. Because of the application simplicity, such operating system services are very preliminary. Therefore, it is not preferable to use them in WSNs, because WSNs' applications are more complex than those in such systems.
- On the other hand, on general purpose embedded devices such as PDAs or some smart cell phones, small versions of traditional operating systems are used. For example, PalmOS, WinCE, Symbian [15] (Nokia Operating System) or

Embedded Linux. These general operating systems have some restrictions to be used for sensor nodes. For instance:

- Heavy weight process is used in such systems, which the sensor node can't afford due to its limited resources.
- They do not scale down to small devices such as sensor nodes. Currently, storage in the average type of sensor node is in the range of 4 kilo Bytes of RAM and 64 kilo Bytes of ROM.
- Such systems have high energy consumption, because most of these operating systems are designed to work for a limited time (days, weeks) with their dedicated power resources. Sensor nodes are supposed to work for months or even years.

In fact, operating systems for sensor nodes lie between the above two categories and require special kind of operating systems for many reasons such as resource management, modularity, fault tolerance etc. However, it is still considered that, an extra functionality is not desired on these constrained devices because of their several existing limitations. For that we can say that sensor network require special kind of operating systems.

2.1. Design Methodologies of WSN Operating Systems

Operating systems for wireless sensor networks vary in many aspects. We can classify these differences according to the way which is followed in the design:

2.1.1. Control Loop

Some of the operating systems (firmware) are based on the simple main control loop. In this control loop, the execution is suspended until an event occurs. Then, a response will accordingly be signaled. This way of design has a lot of busy waiting but it is the usual way of implementing software applications for industrial embedded systems. We can find such an example in the ESB sensor node [16]. In ESB, the simple resource management functions (firmware) are statically linked to the application in order to form the main executable program. Then, the program is loaded as a whole into the sensor node.

2.1.2. Event-based OS

Another way to build the software for sensor nodes is according to the monolithic events. In this way, the software for managing the resources consists of a collection of event handlers that respond to the events generated in the platform. An example of this type is “PalOS” [17].

2.1.3. Legacy OS

The way of designing the conventional operating systems could also be used in WSNs. Such systems use the TCP/IP protocol and sockets in the network stack. Some designers prefer this way, because of the many ubiquitous devices that use the standard TCP/IP and sockets. This way of implementation is followed by Adam Dunkels in the Contiki Desktop operating system [18]. This system has native μ TCP/IP stack, preemptive multithreading options, event-driven kernel and GUI support for locally connected terminal or through VNC over telnet. The drawback of using such operating systems in WSNs is that, this approach is fairly restricted because WSNs platforms can not afford all functionality of the TCP/IP stack.

2.1.4. TinyOS-type of OS

The final approach of designing the operating systems for the wireless sensor nodes is to provide firmware for concurrency and modularity, which does not block or poll the execution. It gives an appropriate abstraction and handles the resources in optimal way. This way of implementations is followed at the University of Berkeley in the operating system named “TinyOS” [10].

2.2. Design Considerations of Operating Systems for WSNs

Few aspects that must be considered while designing an operating system for sensor nodes are described in this section. This description presents properties and requirements in the WSN operating systems as they are distinct from the conventional operating systems. One should consider the constraints and the limitations of the processing,

memory, energy and communication capabilities. These and other considerations are described in the following paragraphs.

2.2.1. Programming Languages

In this area of the research, the compiler can provide a big support in the design. The research is going on to put the compiler in charge of most of operating system duties. Choosing a suitable programming language (assembly or high level language) that adopts the nature of the hardware is a good step.

Every microcontroller family has its assembler or has high level compilers like C or Java. High level compilers compile either to the assembly code or directly to binary code. For example, AVR from Atmel has the compiler `avr-gcc`; MSP from TI (Texas Instrument) has the compiler `msp-gcc`.

A comparison between C and nesC is here made; the two most often used programming languages for sensor nodes. C language provides multiple low levels for accessing the hardware, but it provides less help in writing safe code and structuring applications. On the other hand, nesC solves this problem by using components. C compilers are provided for most of the microprocessors and they are provided with a tool chain which supports the development, while nesC is only ported to few microprocessors. A tool chain normally includes cross compiler and bin-utils (assembler, linker, debugger ...). nesC provides context model for concurrency, reusability and prevention of race conditions which C does not provide. Hence, compilers with advanced compiler techniques provide a great support for operating systems and applications development of sensor nodes.

2.2.2. System Resources Management

Adopting classical schemes for resource management of WSNs would not be the right choice as sensor nodes have very limited resources. Therefore, special efficient operating system schemes are required in designing the system modules such as file system, networking stack and CPU scheduling.

2.2.3. Real-Time Requirement

In some WSNs applications, real time is the most important aspect and no deadline must be missed. For example, the controller of an aircraft missing a deadline could lead to a crash. However, the focus now is rather on algorithms and flexibility than on real time characteristics. After finding efficient algorithms, the system can be further optimized to fulfill the real time requirements.

2.2.4. Application Specificity

Wireless sensor networks are application-specific networks because of many reasons. There is a wide range of applications in which we can employ wireless sensor networks. Processors in the sensor nodes are very specific and might have variety of modules such as UART, USART, TIMER, and SPI. They might also support many communication methods such as RFM, Ultrasound Bluetooth, or IrDA.

Some researchers propose to have, therefore, higher specific layer in OS as virtual machine to provide a generic computing environment for general applications but we are still far away of accomplishing generic sensor node platform.

2.2.5. Algorithms Independency

Algorithms should be as much independent as possible from the platforms. This will provide portability between different types of platforms.

2.2.6. Process, Threads and Scheduling

Single process system is the usual case in embedded system applications as well as in the WSN nodes. However, the availability of multiple threads makes the system more controllable than the single process system.

2.3. Sensor Node Hardware

Today, there are many projects dealing with wireless sensor networks. The oldest project was Smart Dust (Autonomous sensing and communication in cubic millimeters) [19]. It was finished in 2001, and most of the largest research groups developed out of the Smart Dust project, for example, Nest (Networked Embedded System Technology) [20], CENS (Center for Embedded System and Networking Sensing) [21], Berkeley WEBS (Wireless Embedded System) [22].

Most of these research groups do not design special hardware components or elements for their platforms. But instead, they use Commercial or Common off the Shelf components (COTS). Moreover, they deal with diverse existing elements such as many microcontrollers (AVR, MSP etc.), different types of sensors (humidity, temperature, light) and multiple types of communication devices (IrDA, Radio, Bluetooth,...). However, very few groups do not use COTS. Instead, they are developing their hardware through the Co-design of hardware and software, such as using FPGA technology for getting higher and faster functionality. An example for that is the Tyndall Mote sensor node at the Tyndall Institute [77]. The variation of the hardware components has provided multiple types of wireless sensor nodes. Moreover, the quick development in these devices resulted in multiple generations of wireless sensor nodes. The following are some of the current famous sensor node platforms.

2.3.1. MICA Series

Many sensor node generations have been designed at the University UC-Berkeley (with Crossbow [9]) for example RF-Mote (1998), WeC (1999), Rene, Rene2, Mica, Mica128 (2001), Mica2 (2002), Mica2Dot, MicaZ, Telos (2004), Tmote and Invent (2006).

Changes are mostly in the type of the processor and communication components. Every generation has replaced the processor with some other processors. For example, the following processors were replaced sequentially AT9080515, AT90LS8535, ATmega163, ATmega103, ATmega128l, and MSP430. Similarly, in the radio devices the following TR1000, CC1000 and CC2420 came after the other. Figure 2.1 shows a representation of a sensor node platform from Berkeley nodes [23]. One sees in this

figure the microcontroller and its main peripherals and the multiple components connected to it.

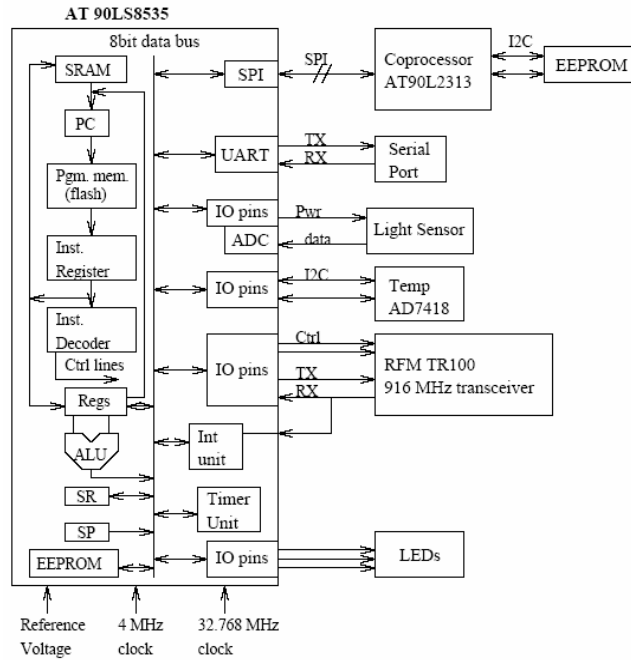


Figure 2.1 Representation of MICA wireless sensor node platform

2.3.2. BTNodes Series

In the framework of the Smart-ITs project [24] at the ETH Zurich, a different and well-accepted class of sensor nodes has been designed. The designed nodes are BTNode1, BTNode2, and BTNode3. The components used in these nodes are: ATmega1281 processor, an Ericsson ROK 101 007 Bluetooth module, Chipcon CC1000 and Zeevo ZV4002. Figure 2.2 shows the nodes from ETH.



Figure 2.2 BTNodes in ETH Zurich

2.3.3. ESB Series

The CST group at FU Berlin has introduced many generations of its platforms ESB (Embedded Sensor Board), ESB2, and ECR (Embedded Chip Radio) [16]. They have used in the design MSP430F149 from Texas Instruments. For radio, they use the transceivers TR1000 and TR1001. Their new ECR sensor node adds higher scaling support for the network. Figure 2.3 shows the ESB and ECR:



Figure 2.3 ECR, ESB nodes at the FU Berlin, CST group

2.3.4. Sun Sensor Nodes

Researchers at Sun Labs have developed their own sensor node hardware platforms [25]. Sun nodes, Figure 2.4, use 32 bit ARM-7 CPU and 2.4GHz radio compliant to IEEE 802.15.4. Sun nodes are relatively strong nodes because of the powerful microcontroller which is 180 MHz, and has relatively large memory (512 KB RAM/4M KB Flash ROM). However, this restricts Sun nodes from running for longer time, thus limiting their suitability to only short term applications.

The Sun nodes are based on Java technology. To develop applications for Sun nodes, developers can use any JAVA Integrated Development Environment (IDE) which supports the JAVA compiler and the debugger. By using Java, one can also benefit from the portability of Java in order to provide multiple platforms with the same high level software applications.

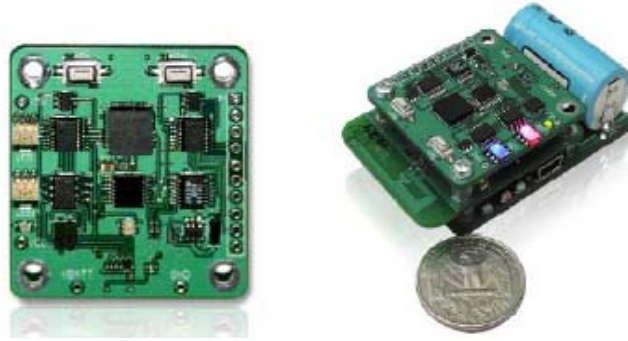


Figure 2.4 Sun nodes

2.3.5. MarathonNet Sensor Nodes (PaceMate)

Interesting Project, in ITM group [26] at the University of Lübeck, Germany, has introduced a new platform called PaceMate. PaceMate, shown in Figure 2.5, has been applied in the project MarathonNet. In MarathonNet, vital signs of athletes are sensed and then sent to the surveillance stations during the marathon event. The notable feature of this platform is that it has a display instead of having the handy LEDs. This display is used to show the user some of the sensed parameters such as the athlete's position, others' positions and the vital signs values during the race.



Figure 2.5 Pacemate sensor node

2.3.6. iSense Sensor Nodes

iSense sensor nodes [26], Figure 2.6, are market-competitive sensor nodes. Their designers have exploited the learnt lessons of the long research process carried out on wireless sensor networks at ITM (at the University of Lübeck) [26]. iSense sensor nodes have a general module, to which the user can attach custom sensor modules depending on the application requirements. Three types of sensors modules are foreseen. First one is for

tracking environmental parameters. The second is for movement detection, and the third is for sensing the passing objects.



Figure 2.6 iSense sensor node

2.4. nesC

nesC stands for Component oriented language for Networked Embedded Systems. nesC language is explicitly designed to be used for WSNs. nesC provides the model for writing both the operating system components and the application components for a sensor node. nesC is an extension of C. Furthermore, nesC compiler generates from the nesC source code the C source code, which can be finally compiled with a standard C-compiler into assembly or directly to binary code.

In the next subsection, nesC features are briefly discussed.

2.4.1. Characteristics of nesC

Figure 2.7 shows the compilation process followed in compiling from nesC to C and then to executable files.

In the compilation, the cross compiler and “binutils” (bin utilities) package are used. “binutils” provides the needed utilities in building the object file. Some of these are:

- avr-as assembler.
- avr-ld linker.
- avr-ar librarian.
- avr-objdump disassembles information from object files.
- avr-strip strips information from object files.
- avr-objcopy extracts information form object file.

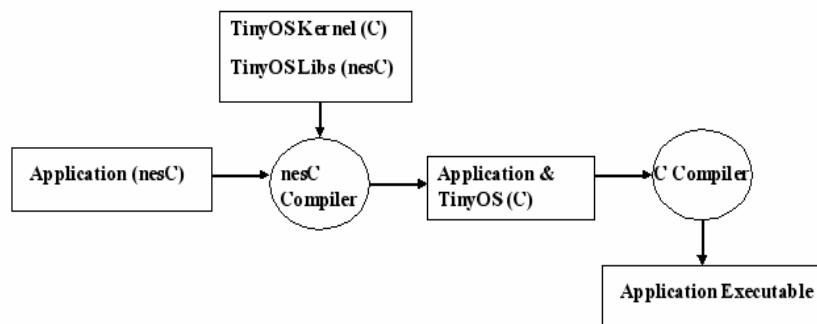


Figure 2.7 Compilation operations from nesC to binary file

The five main concepts of nesC language [42], especially as compared to C, are summarized in the following.

- **Independence of Composition and Construction**

nesC provides high abstract modules that are assembled to form the final binary code. These modules are of two types: Implementation components (called modules), which provide the actual implementation of the functionality, and Configuration components used for merging another Configurations or Implementation components.

- **Components Interfaces**

Interfaces represent the standard specifications that each component provides or uses. The interfaces can be used or provided. The component which provides an interface must provide the implementation of the functionality inside that interface. On the other hand, the component which uses an interface can use any of its implemented functionality which is offered by the interface's provider.

- **Interfaces are Bi-directional**

The component that implements an interface can concurrently call a call back functions or event. This feature enables non-blocking operations. For example, in lengthy operations such as packet sending, the system switches to perform other activities during the sending. Sending completion is signaled through an event "sendDone" in the bi-directional interface.

- **Static Linking**

In nesC, no dynamic linking occurs on the nodes while operating and all components are statically linked together via their interfaces. This way, the right behavior of the program is guaranteed, as memory requirement is decided while compilation. Moreover, this will also increase the runtime efficiency.

- **Concurrency Model**

nesC supports static abstract code units (called tasks) which can be scheduled during the run time. These units run till completion in the current compiler and only can be preempted by interrupt handlers. To ensure race-condition-free code, the compiler signals with all potential races during the compilation and user should fix that before loading the program to the sensor node.

2.4.2. Code Optimization in Wireless Sensor Nodes

There are today many small dynamic memory allocators that fit in such small nodes, but there are still difficulties that might be caused by the dynamic allocation such as the large overhead and memory errors.

Operating system in the sensor node has one process which has all code segments as in an ordinary address space (.heap, .bss, .text, .data). Segments allocation of the process can be done by using the linker, which decides about where to place all the segments (.data, .bss and .heap).

To increase the quality of the code and optimize the resource usage, the designer of an application for WSNs should follow a number of rules. For example, using in-lining functions reduces the overhead of calling these functions. Busy-waiting and spinning should be avoided because it decreases the efficiency. It is also very essential to know the storage requirements at the compile time.

2.5. *TinyOS (TinyOS multithreading Operating system)*

TinyOS operating system initiates a new approach for managing the resources in WSN nodes. TinyOS is an open source operating system, well-known for its wide usage for WSNs. It is designed using nesC.

Out of the investigated operating system, which are designed for WSNs, such as TinyOS [10], firmware for ESB sensor nodes [16], Contiki Operating System [18] and MANTIS OS [27], mainly for our work, we have selected the TinyOS operating system for many reasons. Some of them are the following:

- It has already been ported to function with almost all sensor node hardware types (Moteiv [12] node series, BTNode series [24], ESB [16] sensor nodes, iSense nodes [26] and others).
- TinyOS is supported and developed by a large community, which increases its reliability.
- TinyOS provides flexible management to the limited hardware and software resources. This system also provides a stable consistent abstraction to the application developers to interact with the hardware without having to know about the details of the hardware.
- TinyOS efficiently works under bursts of concurrency intensive operations which are generated either from the sensor duties or from the network duties.
- Power efficiency is one of the main features supported by TinyOS.
- TinyOS supports efficient modularity to design complex applications; so that the designer can easily compose the operating system and applications without much difficulty.
- There are many existing libraries; one can use them for building the application.

In this operating system, with support of the nesC compiler, components are the main parts. In the following, TinyOS's structure, components types and their structures are briefly explained.

2.5.1. TinyOS Structure

Most of the operating systems designed for WSNs have similar hierarchical layered structure. They vary, however, in the logic of handling the functionality inside the layers. The structure of TinyOS operating system is shown in Figure 2.8 (Round shapes represent components). The layers shown in this figure are:

- **Main component:**

This is the top level of the operating system. It holds the code related to the (main ()) function body that has to initialize hardware, and then keeps on scheduling tasks. In the next illustration, we include a pseudo-code of the main nesC, as following:

```
int main() __attribute__ (C, spontaneous){  
    call hardwareInit();  
    call Pot.init(10);  
    TOSH_sched_init();  
    call StdControl.init();  
    call StdControl.start();  
    __nesc_enable_interrupt();  
    while(1) {  
        TOSH_run_task();  
    }  
}
```

In the above code, hardware components are firstly initialized. Next, the potentiometer starts controlling the power. After that, the scheduler is initiated to start scheduling the jobs. Later on, in the while loop, threads are being executed.

- **Application components:**

This layer contains components defined by the application designer and it is designed according to the application nature. An example for implementation of this layer is Tiny DB which is a query processing application for extracting information from a network of sensor nodes [10].

- **Hardware independent layer:**

These components implement services offered to the application components. They also provide the reusability of the system components for different applications.

- **Hardware presentation layer (HPL):**

It abstracts all functions from the hardware into the low system components that wrap the underlying hardware. These groups of components are the most dependent ones on the platform. After taking the abstractions from all the parts in the hardware, designers assemble them in optimal way to provide the useful functionality. For example, “UART.nc” component is used as a hardware wrapper. After its initialization, it offers all commands and events (hardware interrupt) taken from the hardware which deal with UART port. The system uses these provided functions and merges them with other components.

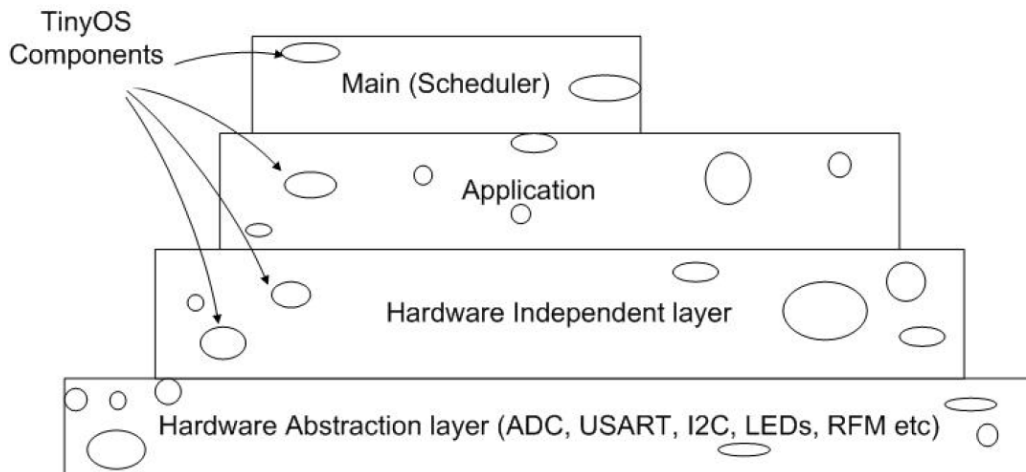


Figure 2.8 TinyOS structure composed of a scheduler and a graph of components

2.5.2. Components in TinyOS

In TinyOS, components are of two types: Modules or Configurations. In the following, an example is taken to explain how modules and configurations are generally implemented.

2.5.2.1. Modules

Modules are used for implementing the provided interfaces. This implementation can use other already implemented interfaces. The following pseudo-code represents an implementation for this module.

```

module App {
    uses interface Comm1, Comm2; // Used Interfaces
    provides interface Init; // Provided Interfaces
}
implementation { // Here, it comes the actual implementation
    int sum = 0
    command void Init.init() { // Implementing a Command
        call Comm1.readSensor(); // Using a Command
    }
    event void Comm1.Send(int val) { // Implementing an Event
        sum += val;
        call Comm2.readSensor(); // Using a Command of an Interface
    }
    event void Comm2.Send(int val) { // Implementing a Command
        sum += val;
    }
}

```

In this code, first with “module” keyword, the name of the module is defined. Then, “implementation” keyword contains implementation of the provided commands and the used events supported by this module.

2.5.2.2. Configuration

The second type of components is called configuration. Configurations wire modules or/and other configurations. Configurations are used to form the top level of the application. Every application must have wiring components (Configurations) and implementation components (Modules). The following pseudo-code represents a configuration.

```

configuration AppC { }
implementation {
    components Main, App, CommC; // Components inside the configuration
}

```

```

Main.Init -> App.Init;      // Wiring two components
App.Comm1 -> CommC.Temp;   // Wiring two components
App.Comm2 -> CommC.Light;  // Wiring two components
}

```

“Configuration” keyword is used to define the name of this configuration component, while implementation keyword is used to provide its functionality. The symbol “->” is used to bind used interfaces (on left side) with provided interfaces (on right side).

2.5.3. The inner Structure of a Component

A component, as shown in Figure 2.9 [23], has four interrelated parts:

- Command handlers: these are commands implemented inside the component.
- Event handlers: these are events triggered by another component.
- State frame: this contains the state of a component.
- Tasks: the tasks represent the pieces of code whose execution can be deferred.

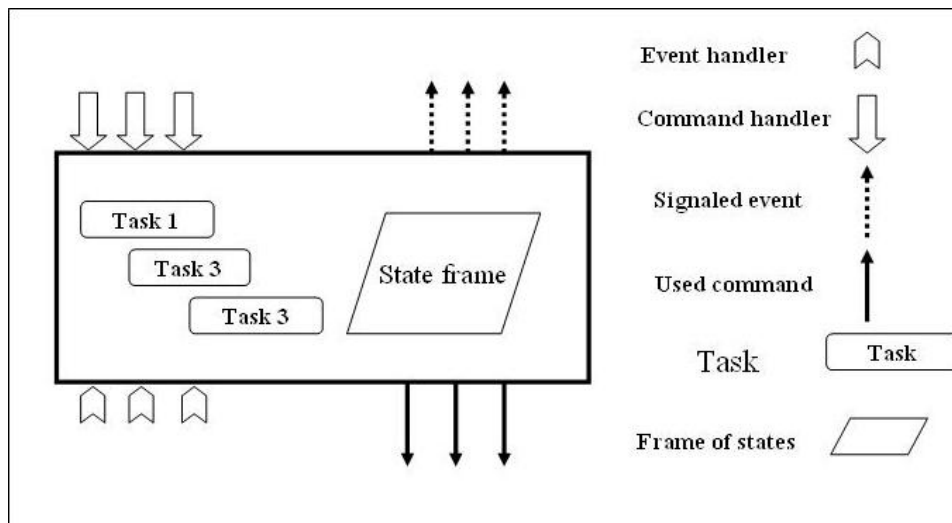


Figure 2.9 Component architecture

2.6. Energy Consumption in Wireless Sensor Networks

Nodes in wireless sensor networks have very limited energy resource. Therefore, all system modules should be power-efficient. In the course of the design process of an operating system or applications for a sensor node, some rules must be considered for conserving the power. For example, the system must never poll. Instead, it must be interrupt-driven, because polling draws resources and consumes a significant amount of the energy.

In addition, software efficiency is achieved by the reduction of the CPU clock. Software efficiency can be translated into power-saving. Increasing the code efficiency gives longer life time to the nodes.

Today's hardware architectures of sensor nodes offer many efficient power saving mechanisms. For example, old processors typically have two function modes, the sleep and the wake-up mode. Recent processors used in WSNs have 6 different modes (Idle, ADC noise reduction, power down, power save, standby, extended power save). These modes allow the designers to include efficient power consumption in their design.

Due to the high cost of communication, it is considered as the most important factor in this regard. Normally, the energy required for sending one bit over the wireless medium is in the range of 1000 to 10,000 times of the energy required for processing that bit [78].

Two of the most commonly used transceivers are the RFM TR 1000 and the Chipcon CC1000. Both have multiple energy control modes. The transceiver chip CC2024, from Chipcon Corporation, has in its control registers a number of bits to control the reception and the transmission power, the frequency synthesizer and the crystal oscillator. This control enables the application designer to optimize the system for low power consumption.

In routing schemes for WSNs, the routing algorithm must ensure efficient energy consumption while choosing the routes, no matter what the algorithm is a link-state or distance-vector based. Also, the system should be capable of short multi-hop routing instead of increasing transmission strength, because, in case of high transmission strength, the energy consumption for sending data will be very high comparing to short mutli-hops [28].

TinyOS is an example of the power-efficient systems. It uses special component responsible for power management. This component has an interface that contains a command for sending the system to sleep mode and an event for waking the system up. This component restores the states of all the microcontroller's ports and registers, before sending the system in the sleep mode, and then it returns the old states of the system while waking up.

2.7. Networking the Nodes in Wireless Sensor Network

In WSNs, the nodes coordinate with each other to achieve assigned tasks. Robust, efficient and flexible communication schemes should be provided in WSNs due to its scarce resources. In this section, it is explained why the classical model of communication stacks are not used in WSNs nodes.

2.7.1. Networking Stack Characteristics

Sensor nodes do not have a separate network adapter or even a separate network processor. The main processor is involved in every single byte to be received or sent. In WSNs, there are many restrictions in setting legacy network protocols (TCP/IP). Some of them are:

- Internet protocol suite (TCP/IP) depends on reliable link approaches at the lower levels, which leads to more energy consumption. Such legacy communication protocols are also optimized for high bandwidth as available in today's networks. However, in communication in sensor nodes, bandwidth is typically limited to the range of few Kbps.
- In Internet protocol suite (TCP/IP), the sender waits until the receiver sends the acknowledgment. This guarantees reliable transfer but it is not flexible because a large amount of the consumed bandwidth due to delays, acknowledgments and retransmissions. However, in case of wireless sensor networks, some level of packet drops is allowed. This packet drop depends on the application.
- Packets in Internet protocol suite (TCP/IP) contain large headers. This causes high overhead at the end points. Such high overhead can not be afforded by the resource-limited sensor nodes.
- Sockets in Internet protocol suite (TCP/IP) do not fit in such constrained nodes.
- In WSNs nodes, there is no support for parallelism which is needed by TCP/IP.
- TCP/IP provides the best performance with the multithreaded hardware architecture, which is not supported by the current sensor nodes.

What is used instead?

Special communication schemes are required in WSNs. In TinyOS, a generic network stack is provided. This network stack implements the communication over the serial port (UART) and the radio. It also has a special middle layer which is called the “Active Message” (AM) layer. This AM layer provides an efficient communication model which is suitable for the restricted communication capabilities in WSNs. In the following section this model is described.

2.7.2. Active Message Model (AM) in TinyOS

AM communication model is originally used in parallel computing. It is used in sensor nodes due to the similar event-based nature of the operations in wireless sensor nodes and parallel computing. The AM model is an asynchronous communication model, which combines the computation and the communication in the node. Under this model, the network can be seen as a pipeline operating at a rate determined by communication overhead. Therefore, AM reduces the amount of the needed buffering, which cannot be tolerated by the physical hardware of the sensor node. This model also provides quick and asynchronous handling of messages. Figure 2.10 shows the difference between the traditional communication model and the AM model [29]. In traditional communication model, the computation is suspended during data transmission, while it works parallel in the AM model.

In this model there are three aspects provided:

- best effort transmission
- addressing
- dispatching to other layers

Messages are propagated to the application layer from and into AM layer through temporary buffers, which hold the messages until the applications are ready to deal with them. These messages contain the payload and the name of the interrupt handler at the user and the application level.

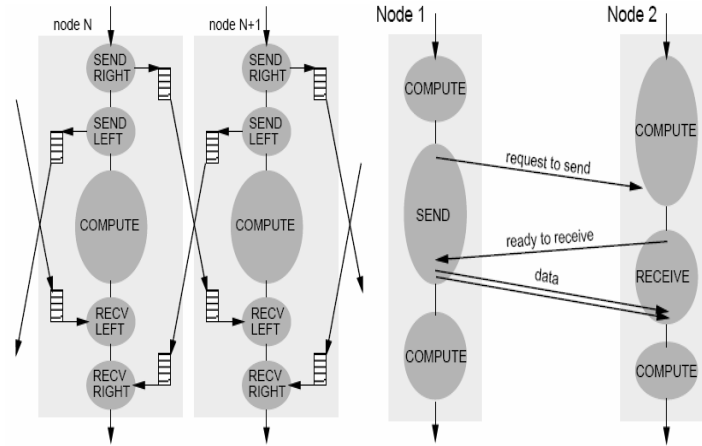


Figure 2.10 AM active message model (left), traditional communication model (right)

Although the AM layer represents the communication model, it only provides the main functions for communication. If the application needs specific functionality, then that is the responsibility of the application to provide their specific functionality. This reduces the size of the operating system and enables the application developers to define more custom levels for detecting the errors and correcting them through the horizontal communication between the layers.

2.7.3. Efficient MAC Layers for Sensor Nodes

MAC layer in WSNs varies in many aspects from the traditional MAC layer of the voice and data wireless networks. In traditional networks, which are based on the standards such as WI-FI (IEEE 802.11), cellular systems (GSM), Bluetooth (IEEE 802.15.1), the main focus is on Qos rather than the energy consumption, which is of secondary importance.

In WSNs, both contention-based media access protocols such as Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA), and contention-free media access protocols such as Time Division Multiple Access (TDMA) and Frequency Division Multiple Access (FDMA) are used for the MAC layer. Moreover, contention-free protocols conserve more energy because the chance of collisions and idle listening is lesser [31]. However, in TDMA (Time Division Multiple Access), it is difficult to change the frame length in case of frequent changes in the topology. Furthermore, TDMA suffers from the interference issues. FDMA (Frequency Division Multiple Access) is another method that avoids the interference by adjusting each two nodes to different frequencies. The MAC layer algorithm plays a main role of the lifetime of the sensor node. Therefore, main issues should be considered while designing MAC:

- Avoiding idle listening.
- Avoiding collisions.
- Avoiding overhearing
- Minimum control packets overhead.

While designing MAC layer for WSNs, one should also consider the fact that such networks are mostly large scale networks. Collision avoidance should also be optimized to have energy-efficient MAC. Although fairness, latency and throughput are in the secondary level of consideration, they should not be ignored.

There are different MAC layers which are explicitly for WSNs, for example, S_MAC [31]. In S_MAC nodes are in sleep mode most of time. If they detect a change in the environment that should be monitored, they wake up and initiate the suitable event handler, which is responsible for serving the sensing and monitoring. Nodes stay awake till the completion of the handling, and go back to sleep mode. The procedure that is

followed for serving nodes in S_MAC is to assign slots. Within these slots the nodes wake up and listen to the medium and go back to the sleep mode (as shown in Figure 2.11) [31]. Maintaining the synchronization is required to let the neighbor nodes listen and sleep in dictated periods.



Figure 2.11 Periodic listen and sleep

B-MAC [74] is a carrier sense media access protocol for WSNs. It enhances a flexible interface to get low power operations, collision avoidance and high channel utilization. B-MAC supports an adaptive preamble sampling scheme to reduce duty cycle and have less idle listening.

B-MAC has been investigated on real sensor network used for surveillance application. The verification process was done by varying many parameters to see the optimal values for that protocol and also to compare these values with other MAC protocols such as S-MAC. The following variations have been compared: (normal B-MAC), (B-MAC with ACK), (B-MAC with RTS-CTS), (S-MAC Uni-cast and S-MAC broadcast). The verification processes were based on either log files, which can be read offline after the experiment, or monolithic programs on the base stations to observe the monitored parameters. However, by using our designed management application, the verification process would be much easier, faster and more efficient.

Chapter 3

CPU-Scheduling in WSNs' Operating Systems

CPU Scheduling controls the CPU utilization by speeding up, down or suspending the flow of the execution and prioritizing some execution units over others.

In WSN area, the researchers encounter many difficulties in setting advanced CPU-scheduling algorithms due to the hardware limitation. However, research is progressing in parallel with the hardware development. This implies that future designs will surely support all kinds of scheduling methods. In WSN, it is been looked for mechanisms that are adaptive to the limited resources and are supporting less energy consumption.

As we have mentioned in the chapter two, most of the operating systems or firmwares for WSNs do not use any kind of CPU-scheduling and a few do use a simple CPU scheduling schemes such as TinyOS.

In this chapter, proposed CPU-scheduling schemes of TinyOS are presented. Then, a new CPU-scheduling scheme, which we have designed for TinyOS, is demonstrated. Finally, the results obtained from it are discussed.

3.1. Scheduling Model in TinyOS

In TinyOS, a cyclic array buffer is used for listing the CPU tasks, which are scheduled by the operating system. The scheduler runs these tasks one by one in a FIFO fashion. Every CPU task is run until its completion and then the next task is initiated. Therefore, a task should not spin or block for a long period of time; otherwise the system performance slows down. These tasks can only be preempted by hardware event handlers.

The following is few rules must be considered in concurrency model of TinyOS. These rules are represented in Figure 3.1, and these rules are:

- Events are asynchronous code. They can signal other events, call commands, or post tasks. They have also the higher priority than tasks but there is not priority among events.
- Commands are similar to simple functions except that they can be called from other components outside that component where they are implemented. Commands can not signal other events to avoid cycles in the code (that is shown Figure 3.1) [32]. Commands call other commands and post tasks. If commands are (*async*), then they can only be called from an event handler.
- Tasks have less priority than events. Tasks must not run for a long time. They can also signal events or call commands. Tasks can not be preempted by themselves but they can be preempted by other events.

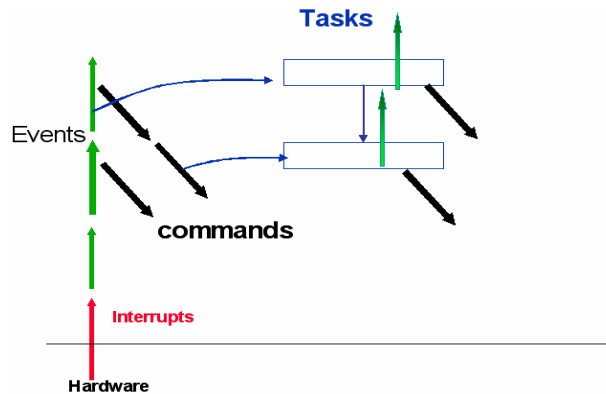


Figure 3.1 The CPU-Scheduling priority model in TinyOS

3.2. Priority Scheduling in TinyOS

In “Priority Scheduling in TinyOS” [7], a further improvement on the simple CPU-scheduling is proposed. This case study illustrates the advantages of setting a priority CPU-scheduling scheme in TinyOS. Consider a high rate, dense and ad-hoc WSN application for aggregating the sensing information and sending them to the base station. In the nodes, the types of tasks (threads) in every node depend on whether to send the raw data or to process it (encrypting, compressing ...) before sending. In the later case, these tasks (threads) can be classified into three types:

- Receiving and forwarding packets to other nodes just for routing purposes.
- Sending local packets that are carrying sensing information.

- Processing local data before sending them. The processing here includes encrypting, en-coding and compressing or any other local activities.

All these tasks can also be either local tasks that are generated from local activities, or network tasks that are generated from network activities.

In this described WSN, if the system in a sensor node reaches a point at which the rate of interrupts is higher than the rate of system tasks. Accordingly, the system will not be able to handle additional tasks. This condition leads to overload the system. Therefore, it is required to give the most critical tasks a higher priority. So, if tasks are dropped under the overloading, the system will not be significantly affected.

In the case study [7], it is proven that prioritizing the tasks coming from the network over the other tasks partially solves the system overloading and provides additional advantages that are:

- In need of a heavy local traffic and routing, this prioritizing enables the nodes to run at higher rate and hence it protects the system from overloading and blocking.
- Energy conserving: the highest cost of the energy in the sensor nodes is at the radio. Here, sending one bit consumes 1000 to 10.000 times more energy than processing it [78]; therefore dropping packets means more energy consumption. As this scheme prioritizes the network tasks over the local tasks, this would decrease the chance of dropping the tasks related to the network activities. Hence, it reduces the chance of dropping the packets.
- The tasks that are bound to send and receive acknowledgments are the most critical ones. If the tasks, generated from split-phase operations such as “SendDone” or “receiveDone”, are dropped due to the engaged scheduler, then the whole packet is considered as a dropped one and is retransmitted; this generates more overhead on the network and on the system. Therefore, prioritizing these tasks of network stack over other tasks reduces this problem.

3.3. *The Deadline Scheduler*

Deadline scheduler was implemented by Pankaj G. Sodagam; extending the simple FIFO scheduler of TinyOS with another paradigm of execution depending on the deadline [33]. Threads are executed here according to their deadlines. In this study, the original scheduler was not modified for avoiding disruption of the operating system. The procedure followed here, is to create another component responsible of queuing threads. This component uses deadline scheduling paradigm. Application can be directly wired to this scheduler through its interface “DTask” implemented in nesC, which has a command and an event as follows:

```
interface{  
    async command result_t deposit(int t);  
    event void dexec();  
}
```

Components need to use the command “*deposit()*” to send their tasks (threads) to the scheduler. Whenever a task (thread) gets executed, components receive the signaling event “*dexec()*” informing that the task has been accomplished. First, the scheduler receives the threads and knows their deadlines through receiving deadlines as arguments. It then compares these deadlines with the deadlines of already queued tasks in order to put this thread in the suitable place in the scheduler. After signaling the execution of every task, the deadlines of all the other tasks existing in the scheduler are decremented by (1). This is very important to prevent starvation of some threads that have a long deadline.

The components that provide the interface “DTask” are “Dscheduler.nc” and “DschedulerM.nc”. “Dscheduler.nc” component is the configuration file. It provides parameterized interface of “DTask” and “StdControl”, so that every application can wire to a unique interface and can parameterize its threads separately from another application. Queuing tasks occurs among the applications separately. The next code represents configuration of “Dscheduler.nc”:

```
Configuration{  
    provides interface DTask[uint8_t id];  
    provides interface StdControl ;  
}  
implementation{  
    component DSchedulerM;  
    StdControl=DSchedulerM;  
    DTask=DSchedulerM;  
}
```

3.4. Topology-Sensitive Priority Scheduling

We have proposed a scheme named “Topology-Sensitive Priority Scheduling”. This scheme has been evaluated through our specially designed simulator [8]. This CPU-scheduling algorithm [1], [2], [8] takes routing issues into account to solve overloading in nodes near the base station. In high rate WSN applications, such as medical or industrial applications, the nodes near the base station do not only have to perform forwarding operations, but also to transmit their own local packets. In the priority scheduling algorithm, which is an approach known for solving the overloading problem of simple FIFO CPU-scheduling, it is not guaranteed that such nodes will get fair chance to send their own local packets if network tasks (threads) are prioritized over the local tasks (threads). We have introduced the concept of priority switching out of the priority scheduling approach, so that such nodes can perform local tasks as well. Our proposal increases the fairness of the distributed sensing. Distributed sensing is defined as the number of collected samples per node, distributed over the unit area (1).

$$S = (C / N) / A \quad \dots (1)$$

S: Distributed sensing

C: Collected samples

N: Number of nodes that take part in the sensing

A: Area where the sensor is deployed

The aim, here, is to increase the distributed sensing (S) by increasing the number of the nodes that are able to complete processing and then sending their local packets.

In the designed scheme, CPU-scheduler behaves according to the information gathered from routing components. Priorities of threads (Tasks in TinyOS) are changed for a limited time based on the knowledge about the topology and then set back to the original values.

In our proposal, the priority of CPU-scheduling is dynamically handled for solving unfair distributed sensing, in which sensor nodes in the topology do not issue packets fairly. Here, the priority of tasks is switched between the sensing and network activities

depending on the values in the routing table. In other words, switching of priority will be based on the changes in the depth in the network. By depth it is meant, that how many hops a node requires to reach the base station in the network. It means that base station itself will be in depth zero, its immediate neighbors will be in depth one and so on. The idea is to give those nodes, which are closer to base station, more chance to periodically perform local activities because it is more likely that they have more routing activities than others.

In the CPU-scheduling scheme, every node monitors its depth in the network. If it moves closer to the base station, closer than a certain depth level depending on the network size (two or less in our example scenario), then it switches its local (sensing) tasks to higher priority for a certain part of its duty cycle, named *switchingThreshold*. *switchingThreshold* is a part of the duty cycle time whose duration depends on the amount of local activities in each node and the size of the network. Within this *switchingThreshold* (Figure 3.2), nodes can complete their local computation and transfer required information in the area where they are located to the base station. Then, priority is switched back to the priority of the critical tasks which are in the network stack. The following pseudo code describes the logical steps of the scheme:

```
If (depthInRoutingTable == 1 || depthInRoutingTable == 2) { // Perform the Scheme
    If (( time % dutyCycle ) > switchingThreshold ) {
        networkTasksPriority = 2; // Network tasks have higher priority
        localTasksPriority = 1; // Local tasks have lesser priority
    } else {
        networkTasksPriority = 1; // Network tasks have lesser priority
        localTasksPriority = 2; // Local tasks have higher priority
    }
} else { // Otherwise do nothing
    NOOP (); // Nodes are not overloaded, as they are far away from the base node
}
```

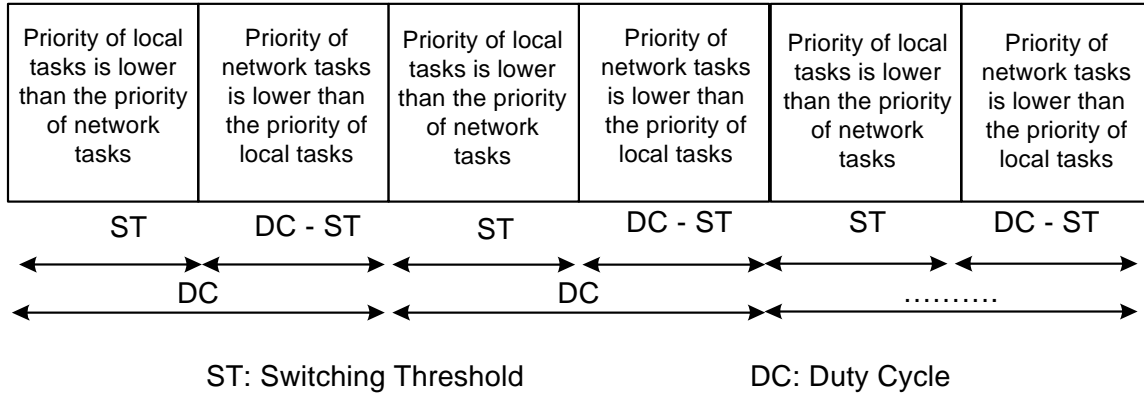


Figure 3.2 The duty cycle for switching priority

For the nodes which are higher in depth than 2, priorities will not be switched as we have implemented that. The scheduler gets the depth by checking the routing tables in the node. As these nodes are not overloaded, switching will never occur unless they change their places in network or the topology is changed. If any of these nodes becomes in the depth one or two, the priority will be switched.

In the nodes, which are in the depth one or two and are not parents of other nodes, they do not have many tasks coming from routing or network stack. Therefore, switching in scheduling will occur, but it will not affect their ability to transfer information to the base station because they are not overloaded.

3.4.1. Results

To demonstrate our protocol, we take an example of a wireless sensor network. This network is used for collecting node samples to a central node “base station”. The base station receives these sensor samples from nodes and also sends commands to nodes to interact with them.

Initially, we have used the simulator “TOSSIM” and its graphical interface “Tinyviz” [34] for simulating our scheme. “TOSSIM” provides simulation option for the TinyOS system components and for most of its available libraries. “TOSSIM” simulates the Mica sensor nodes its networking stack and ADC (Analog to digital converter). It also models the EEPROM at the line level (16-byte block). Furthermore, it can also be customized according to the required parameters of sensor nodes in order to have very fast, scaleable, efficient simulation.

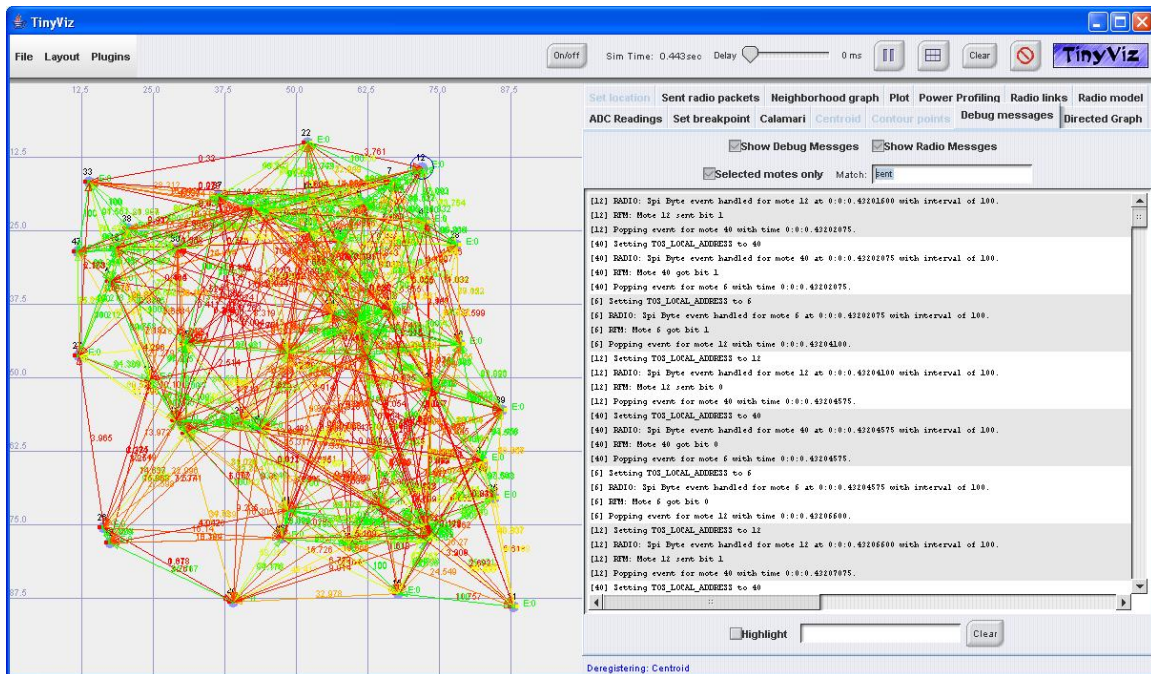


Figure 3.3 Tinyviz simulator

By using “TOSSIM”, shown in Figure 3.3, we have deployed 50 sensor nodes. Each node sends a sample every second. We have noticed that local (sensing) tasks were prioritized over the networking and the routing tasks during the *switchingThreshold* in the nodes that are near the base station. This shows the correct implementation of our scheme.

Although, “TOSSIM” simulator can be used to check the exact behavior of the execution flow and the correctness of the implementation, it is imperfect in simulating the CPU-scheduling schemes because of the following reason. “TOSSIM” schedules the threads in FIFO paradigm just like the scheduler in real sensor nodes. However, there is a critical difference between scheduling in the simulator and in a real sensor node. In real sensor node, the tasks (threads) are pre-empted by interrupts while in “TOSSIM” simulator the tasks are not pre-empted. Instead, interrupts in “TOSSIM” are en-queued in the scheduler in FIFO fashion as tasks. Inside TOSSIM simulator, system will not be overloaded because of that; hence, tasks will not be dropped from the scheduler, which does not correspond the behavior in real a sensor node.

Figure 3.4 shows a simulation in “TOSSIM” of the scheduler in a sensor node following the FIFO CPU-scheduling scheme. We have increased the number of sub nodes that forward through that particular sensor node. The sample rate of all the nodes in the

topology is 1 sample per second. This figure shows the *Throughput* (local and forwarded packets) vs. the number of the increasing *sub nodes*. In this figure, it can be seen that the throughput of this node is not dropping even when the number of sub nodes is increasing, which does not perfectly represent the behavior of a real sensor node, as the node can not send more than 50 packets/second [74]. This proves that TOSSIM does not perfectly simulate the real sensor nodes. This imperfection of “TOSSIM” was the main reason of shifting to our own designed simulator to prove our scheme as it does not en-queue tasks and events in sequential queues; rather, it makes the tasks preemptive.

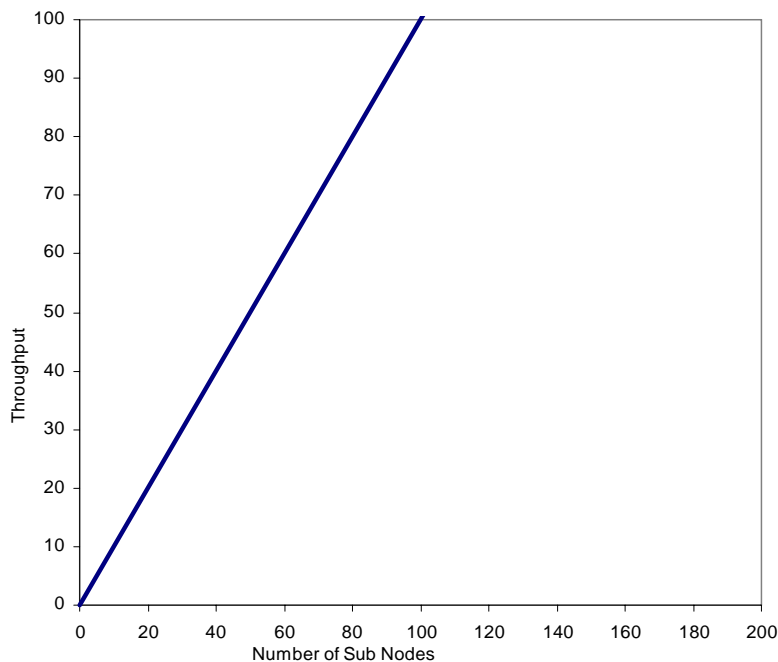


Figure 3.4 Results conducted from TOSSIM simulator for a sensor node that increasingly forwards samples from its sub nodes.

In our experiments on real hardware (Telosb from MoteIV), if nodes run at low sampling rate (less number of forwarded packets), we have observed that the number of packets, forwarded via the nodes closer to the base station, is not affecting the sending of local packets. However, in case the nodes are working at very high sampling rate, the nodes responsible for forwarding are susceptible to system crashes caused by intensive processing operations. These system crashes have prevented from analyzing our scheme

on the real sensor nodes. Due to this problem, we had to limit our analysis just to the simulations performed on our designed simulator.

Our simulator models the execution units as three types:

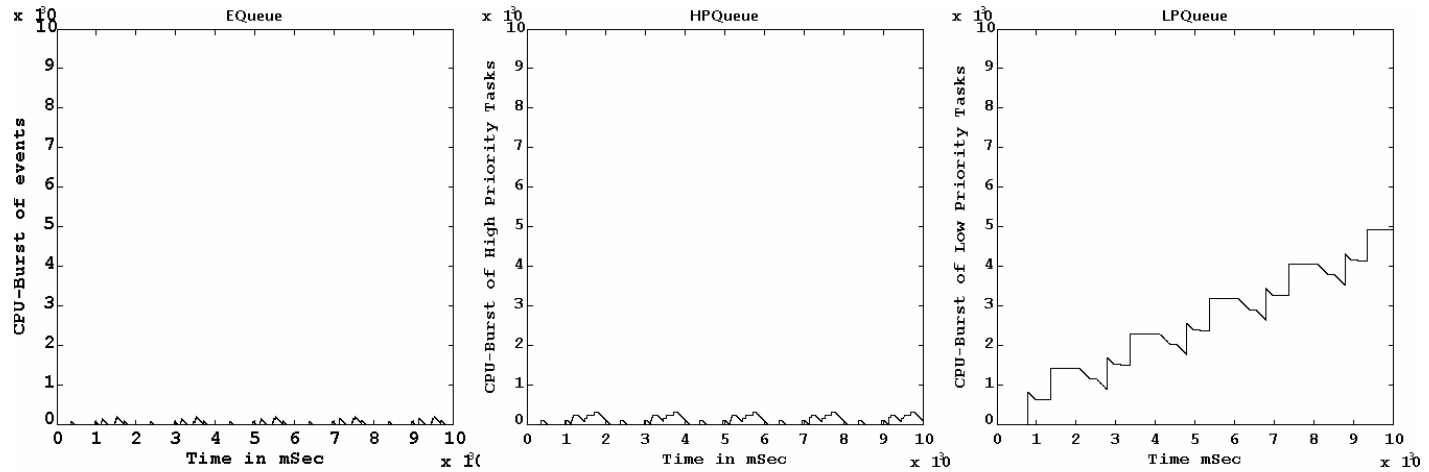
- Events: they are the interrupts executed by the CPU immediately and they always have the highest priority.
- Network tasks (high priority tasks): they have the second highest priority.
- Local tasks (low priority tasks): they have the lowest priority.

Low and high tasks represent the normal processing tasks whose execution can be preempted by the events. These also represent the tasks, whose priority will be switched depending on the position in the topology.

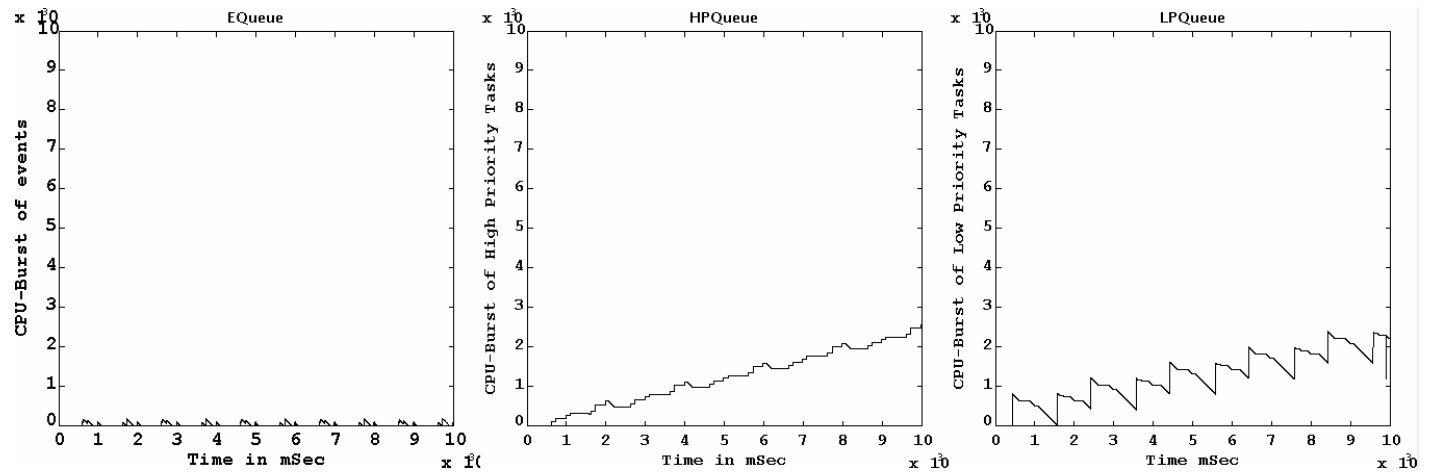
Figure 3.5 shows the simulation of event and task queues of a node before and after applying our CPU-Scheduling scheme. This node is running at sample rate of 0.5 sample/second and it has seven sub nodes which are also running at the same sampling rate. Additionally, this node is at either the level one or two of routing tree, which means it is one or two hops far away from the base station. The simulated queues are:

- EQueue is the event queue
- HPQueue is the network task queue
- LPQueue is the local task queue

Figure 3.5-A shows the task queues performance under normal priority scheduling (without switching). It is observed that there is a high overload at the low priority task queue of the simulated node which causes severe reduction in the local sampling rate. Therefore, the local packets are delayed or dropped by high priority queue. On the other hand, by switching the priority of the scheduled tasks after every 75% of the duty cycle to the low priority queue, we see in Figure 3.5-B that low priority queue is less overloaded. Therefore, the node is able to issue more local packets to the base station. We also notice that the high priority queue is more overloaded which eventually causes a slight drop of network packets on the high priority queues because of the switching. However, it has very small effect as it is distributed on all the sub nodes. One can argue at this point that our scheme is achieving fair throughput of local packets at the cost of forwarding less packets from the sub-nodes.



A: EQueue is the event queue HPQueue is the high priority task queue LPQueue is the low priority task queue



B: EQueue is the event queue - HPQueue is the high priority task queue - LPQueue is the low priority task queue

Figure 3.5 CPU-Scheduler queues of a node forwarding packets for 7 sub-nodes, before and after applying the CPU-scheduling scheme

To evaluate the fairness of our scheme, we have measured the standard deviation of packets received by the base station by increasing the number of sub nodes. In Figure 3.6, the standard deviation at multiple switching levels (without switch 0%, 25%, 50% and 75%) is shown. We can see that the standard deviation in all the variations of switching is always less than the case where there is no switch. It means that having any percentage of switching in scheduling will provide more fairness in terms of distributed sensing. Also, it can be seen that 75% switch is the fairest situation under our assumptions.

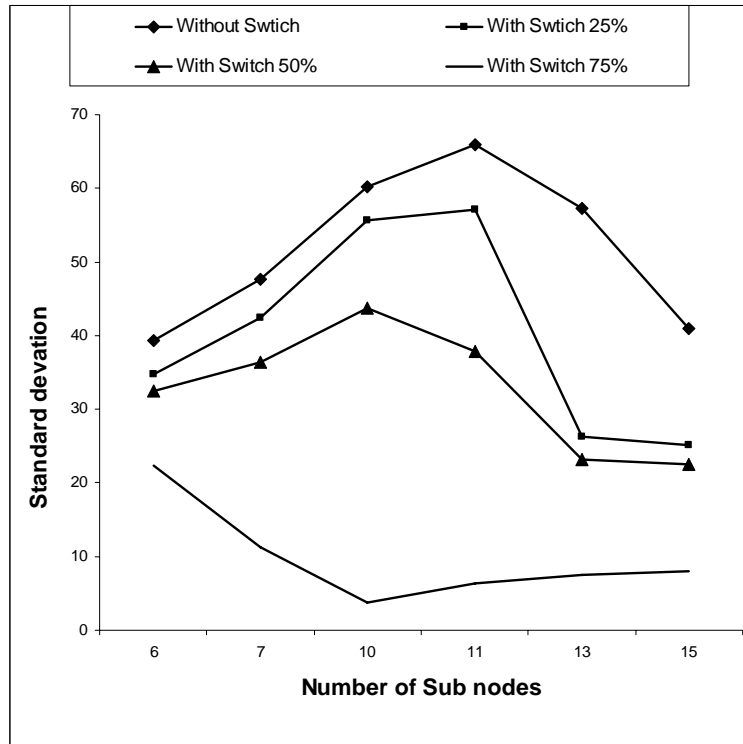


Figure 3.6 Standard deviation of packets received from nodes on the base station

Another important point to discuss over here is what is the optimal value of *switchingThreshold*? To have an answer for this question, we have compared the size of low and high priority queues with different values of *switchingThreshold* for different network sizes. In Figure 3.7, x-axis contains the switching levels (0%, 25%, 50%, 75%, and 100% for q1 and q2 which are the high priority tasks and the low priority tasks respectively) while y-axis represents the CPU-units left in the queue after 100 seconds. Values are taken from a node which has 10, 13 and 15 sub-nodes. It is seen that the load balance between high and low priority task queues can be adjusted through a switching point which will accordingly increase-decrease the number of packets dropping on one of

the queues. The optimal balance value between the two queues requires switching value somewhere between 70 – 80%.

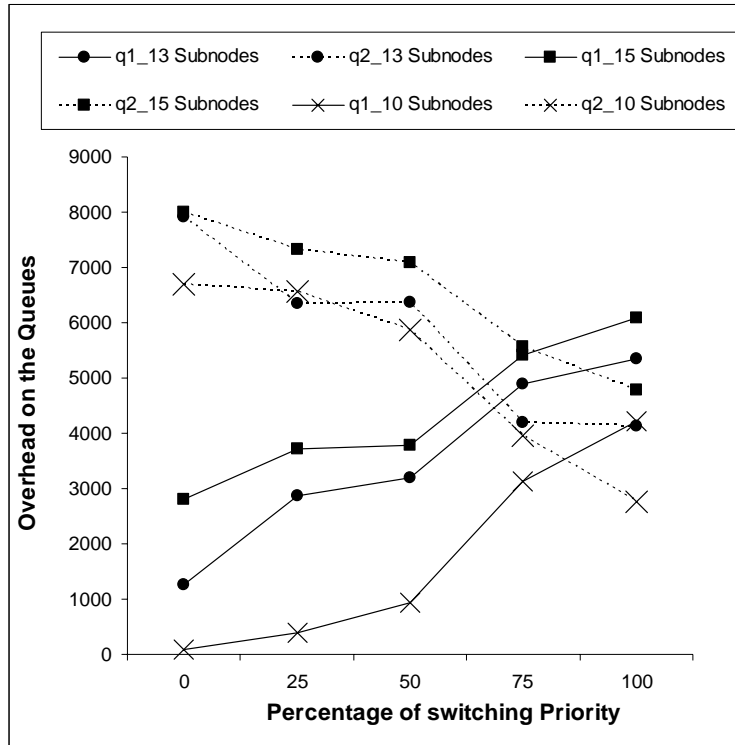


Figure 3.7 Affect of changing the percentage of switching according to number of sub nodes

In conclusion, we have shown that although the priority scheduling provides high throughput and less power consumption in WSNs, it ignores the fairness in the network. We also have shown in our scheme that having different scheduling behavior among the nodes in the topology is more beneficial than applying the normal priority scheduling in all the nodes.

Chapter 4

Wireless Sensor Networks Management

4.1. Definition of Network Management

“The Network Management refers to the activities, methods, procedures, and tools to pertain to operation, administration, maintenance, and provisioning of network systems.” [35].

Performing network management is a complex task. To address this complexity, the management is partitioned into components or layers. There are many partitioning schemes of network management proposed by many international standardization organizations. International Telecommunication Union (ITU-T) has recommended (in recommendations M.3010 [75]) dividing the management into Logical Layered Architecture (LLA) which are:

1. Business management layer
2. Service management layer
3. Network management layer
4. Element management layer
5. Network element layer

From the functionally point of view, network management has five main conceptual areas, as defined by the ITU-T (Recommendation M.3400 [74],[75]). These management functional areas are configuration, accounting, fault, performance and security management. These functional areas are described in the following subsections.

4.1.1. Configuration Management

The configuration management has two main objectives. The first one is monitoring the network so that the user can keep track of the hard- and software parameters. The second is configuring these parameters of the system components such as operating system components, network components and the application components in each node.

4.1.2. Fault Management

The goal of this management is to find out the problems and fix them to prevent the network from malfunctioning. It ensures three main important factors which are reliability, availability and survivability of the system.

4.1.3. Performance Management

The main objective of performance management is to maintain the network so that it runs at acceptable quality level. It also defines the threshold below which the network performance should not drop.

4.1.4. Accounting Management

It regulates the network utilization parameters of the individual groups and it provides fair access to the users.

4.1.5. Security Management

It supports the network with appropriate authorization to the subsystems in the network. It also denies access to unauthorized parts and allows access to authorized network or system parts. In addition, it provides protection against various types of attacks to maintain the overall integrity and the security of the system.

Ruiz et al. [38] have proposed in their “Management Architecture of Wireless Sensor Network” (MANNA) that another dimension should be added to WSNs due to their specific-nature. This additional dimension represents the WSN functionality, as shown in Figure 4.1 [38]. This other dimension has the following layers:

1. Configuration

2. Maintenance
3. Sensing
4. Processing
5. Communication

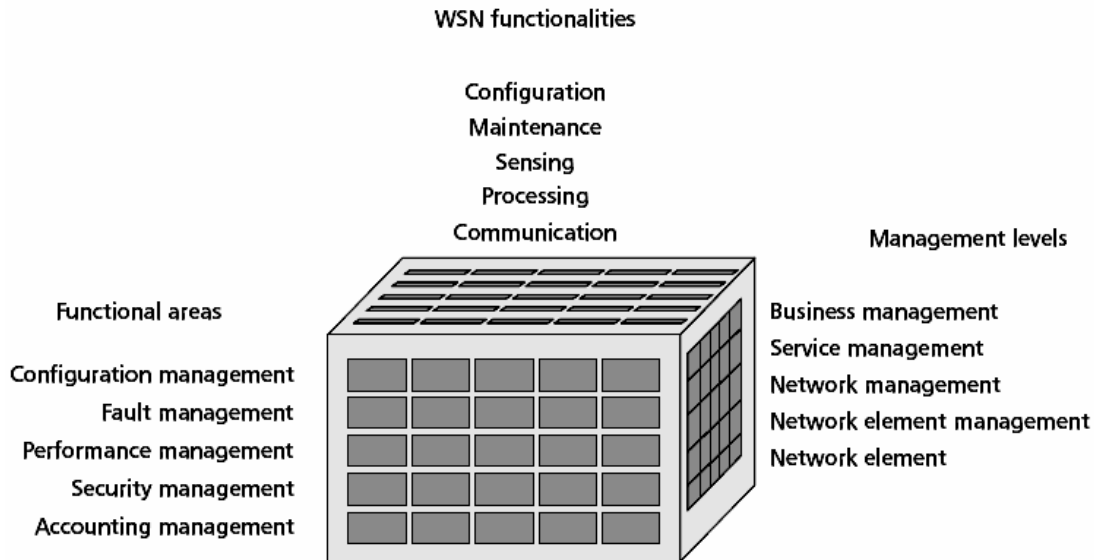


Figure 4.1 MANNA Architecture

4.2. WSN Management Challenges

Traditional network management systems can not be used in WSNs, because of the existing limitations and the special considerations of WSNs. Therefore, the traditional network management solutions should accordingly be adopted. Some of these considerations are:

- WSNs are normally composed of a large number of nodes, which makes managing, controlling and interacting with them a nontrivial task. Therefore, every management solution should consider the high scalability of the WSN. Also, in many applications of WSNs, WSNs are highly dynamic networks because some deployed nodes die or some additional nodes are added or moved to join another sub network.
- Since WSNs have limited energy budget, all the operations, including management services, are expected to be energy efficient.

- Due to the restricted bandwidth, any additional control traffic (management) should be dealt with very carefully.
- In addition, in most of WSNs' applications, WSNs are data centric networks where every node has no unique identification, which is very rare in traditional management network solutions.
- WSN Management protocols are application specific as classified in [65].

4.3. Centralized vs. De-centralized Management in WSNs

In WSN management, normally, there are three potential roles that can be assigned to nodes. These roles are:

- *Monitoring nodes*: these nodes are responsible for, either periodically or reactively, collecting management information and sending them to other central nodes. They also can save these samples in log-files, until queries are received requesting the collected samples.
- *Sink nodes (or sometimes they are called base stations in hierarchical WSNs)*: such nodes are responsible for managing other nodes, aggregating data, forwarding queries or storing data.
- *Gateways*: these nodes connect the WSN to the external entities.

Centralized and de-centralized management can be determined according to the way in which the management traffic flows. In centralized management schemes, most of the time the management traffic flows between monitoring nodes and sink nodes while in decentralized management schemes it flows only among the monitoring nodes.

Most of the researchers, who develop solutions in the field of WSN management, have proposed the existence of either one or more central points (whether it is a base station node, a gateway node, or a cluster head) which manages some other nodes in the network. Moreover, we can generally divide these proposals into three categories based on the way which is followed in implementing the self-management feature. These categories are explained in the following sub sections.

4.3.1. Centralized Management in WSN:

In this management approach, a single or multiple central nodes are responsible for managing the rest of the nodes. This management approach has several advantages such as:

- Firstly, it conserves node resources by performing their complex management tasks and analyzing them on these stronger central nodes.
- The central nodes can store long-terms log-files and statistics and accordingly take advanced decisions on managing the nodes in the network.

On the contrary, this management technique has some drawbacks such as follows:

- Close to the central nodes, heavy traffic is resulted in which limits the network scalability.
- Malfunction some of the central nodes might cause failures in several sections of the network.

In [36], Ramanathan et al. assume that most of the current deployed WSNs and many of future WSNs will be based on data collection applications, in which the data is gathered to central nodes or sinks for analysis. Based on this assumption, they have proposed that the failure diagnoses can be more efficiently analyzed and investigated in the centralized management paradigm. SNMS [14] is an additional example of the centralized management.

4.3.2. De-Centralized Management in WSN:

In this management paradigm, there is no specific node responsible for performing the management of other nodes. In fact, every node can potentially be the manager of its neighboring nodes. On one side, this will enhance the scalability of the network and give better robustness in case of failure. On the contrary, this scheme is difficult to implement and requires a significant part of the nodes resources, especially the memory and energy. MANNA [41] is an example of this de-centralized scheme.

4.3.3. Hyper-Centralized Management in WSN:

In this case, special nodes, such as cluster heads in clustered networks, are responsible for the management of sub-nodes. This scheme is a combination of both centralized and decentralized schemes. For example, it is used in “Scheduling Nodes in Wireless Sensor Network: A Voronoi Approach” [39] and “A topology discovery algorithm for sensor networks with applications to network management” [40]. In [39], the authors assumed that their algorithm, which is to schedule switching nodes off and on in the topology, should run either on the base station or on head clusters.

From the centralization perspective, the aim of designed tool is to provide two categories of management services, which are:

- Managing and configuring of the WSN externally with minimum human intervention: The management services are centralized. This would support the hierarchical WSN for data-collection networks as referred in [36], [37], [63] and others.
- Managing and configuring of the WSN internally without human intervention: This includes the services among the nodes themselves. The management services are thought to be a part of a de-centralized management schemes.

4.4. *Management Models in Wireless Sensor Networks*

Management models provide different information about networks entities, their characteristics and their dynamic changes. In [38], there exists a good classification of the potential models (maps) of WSN management. These models are:

- *Sensing coverage map*: this model provides an image of the nodes distribution, whether there is a sufficient number of nodes at some particular space or whether the nodes were sparsely or densely deployed. This model also presents whether sufficient reporting rate sampling is made or not.
- *Communication coverage map*: it represents the places which are inside the communication range of nodes.
- *Behavioral model*: it studies the deterministic versus statistical and probability models for finding a suitable behavioral models for WSNs.

- *Network topology model*: it models the network structure, for example whether it is flat or hierarchal kind of topology.
- *Residual energy map*: in wireless sensor network, in case of hierarchical collection application, the nodes near central nodes will not only send their packets, but also forward their sub-nodes packets. These nodes lose their energy faster than those nodes which are far away from the central nodes, as shown in our previous scheme. The models which represent differences in energy consumption are also essential, for example in order to find weak places in the networks.
- *Usage standard*: in WSNs applications, some nodes are more likely to be exploited than the others because the often occurrence of some events at particular places. These nodes will be more utilized than the others. A model describing such a usage map would be beneficial for the management.
- *Cost model*: this model is to decide the cost in order to achieve a certain level of management. For example, it measures the equipment and the human cost involved in the management.

Chapter 5

Challenges and Related Work

So far, a lot of research has been conducted in WSNs. However, less attention has been given to the integrated management solutions. WSN management provides many elegant services. However, achieving this management is not a trivial task because of two main reasons. Firstly, WSNs are mostly large scale networks. Secondly, nodes in WSNs have limited resources.

In this chapter, we first explain what could be the management challenges that are faced in WSNs. Then, we elaborate on the motivations and mention the benefits which can be obtained from the designed management tool. Finally, we categorize the related work that has already been accomplished in this area.

5.1. Motivations and Challenges

In WSNs, operations on the sensor nodes are highly concurrent and reactive, making it very difficult to run experiments or develop applications without help of external management tools.

Although, most of the management and controlling tools are only targeted for research purposes, future designs will also involve users because the development WSNs promises users to manage and set up the nodes at their homes, factories or offices through such supporting tools. Following are the challenges which have motivated for designing the management and interaction application for WSNs.

In our proposal, we aimed to deal with all these following issues by designing our management application.

5.1.1. The limited accessibility of WSNs

Inspecting the internal state of programs on the sensor node is very laborious task. There are many existing debugging and inspecting methods in use for WSNs. However, these are very restricted and insufficient. Moreover, the lack of the input/output interaction methods based on human senses limits the accessibility of the sensor nodes in the field. Some of these methods are the following:

1. Using the LEDs: LEDs are used in sensor nodes to indicate that a specific event has occurred such as sending or receiving a packet, triggering a timer, reaching a threshold value, taking a sample etc. This method is very restricted and limited in the deployment scenarios, as the user has to be near every node to observe these LEDs.
2. JTAG connections: using the JTAG connections is an efficient way for inspecting, debugging and monitoring operations on a sensor node. The limitation of this method is that the node must be wired via the JTAG connection in the field, which is infeasible in real life scenarios.
3. Emitting audible noise: This approach is similar to the use of LEDs. But, a sound is here emitted instead of the light in case of the LEDs.
4. Offline logging files: in this approach, management data is saved in log file while the nodes are functioning. Then, these log files are transferred to a PC or a laptop for offline analyzing purposes.

As these methods are very restricted and not inefficient to provide observations on sensor nodes in the field, a new management application is proposed. This application provides efficient management solutions for WSNs without the need for physically interacting with nodes. In addition, it provides observations on the exchanged effects between components. It supports the dynamic remote configurations of the parameters in the nodes. It also displays the results in human-understandable and user-friendly way. These features make the designed tool very beneficial for dealing with the nodes in WSNs.

5.1.2. Frequent Node Failures

Systems of the nodes in WSNs are susceptible to failures like system crash, system freezing etc. In such situations, we either need to restart the system many times -some

times to even stop some of the nodes on functioning- or have to change some parameters on the fly for recovery purposes. This is required especially when we run nodes in critical conditions, like high rate applications or intense density of nodes. The designed management application enables remote recovery of the nodes. In addition, it provides a fault tolerance model to get more accurate sensing values.

5.1.3. Weak Computation and small available Memory

Nodes in WSNs do not provide compiling or code linking operations. It is mainly due to the weak computation and the small available memory. Thus, dynamic configurations are very difficult to be conducted on the nodes. The usual way to perform dynamic configurations is to wirelessly connect the nodes with a strong external computing machine, which can be a special node, PC, PDA etc. This computing machine can take control and process the dynamic reconfiguration, and send the results back to the nodes. On such computing machines, a tool such as our designed tool can perform these dynamic changes.

Furthermore, complex tasks can not be conducted by the nodes. It is due to the weak computation and small available memory. For example, the nodes increase the sampling rate, if the average temperature of the whole monitored area exceeds certain critical threshold during a week. Such a decision based on a long time period can not be taken by the nodes in network. However, management tools running on strong computing machine can take such decisions based on the data collected during the whole week. Afterwards, the tool can respond either by increasing or decreasing the sample rate of the nodes, or can give any other appropriate response. Such tools are also adaptive to the feedbacks or to the collected data from the nodes such as node status or environmental changes.

5.1.4. General WSNs Interfaces

Usually, in WSNs the existence of external entities (PC or PDA) to control the nodes is inevitable. In WSNs, the relation between these external entities and the nodes in the networks can be seen as a Client-Server relationship. Nodes in the network represent the server as they receive and process the queries and the requests coming from the external entities which represent the client. There are many WSNs projects that follow the concept

of having external PCs or base stations, such as MarathonNet [64], CodeBlue [37], and many more.

Our tool exploits this concept by performing difficult tasks on the PC side. It also provides PC-side interfaces, via which users can access the nodes either directly on local PCs or remotely through the external networks such as the internet.

Having general interfaces is a difficult task in WSNs due to the WSNs' application specificity. Moreover, there are a large number of heterogeneous interfaces which could be provided. A proposal that partially addresses or resolves this heterogeneity would be a great support for WSNs. Therefore, we have designed our tool in such a way that its management schemes provide general interfaces to the user.

5.2. Related Work

In WSN, obtaining credible comparison of research results is complex task due to the following reasons:

1. Research results are based on simulations due to the difficulties in dealing with the real sensor nodes during the deployment.
2. Usually, simulations on WSNs are conducted on traditional network simulators such Network Simulator (NS2).
3. Replicating or adopting research results of others' research work is a laborious task due to the specificity of WSNs.
4. Real experiments are mostly performed on small-scale laboratory settings.

W. Louis Lee et al. [66] have surveyed several WSN management research proposals such as MANNA, sNMP, BOSS, MOTE-VIEW etc. These proposals are classified with respect to different parameters such as lightweight operations, robustness, adaptability, storage usage and scalability. Out of the tools in [66], MANNA [38] and Sensor Network Management System (SNMS) [14] are discussed. Furthermore, some of the interaction and management tools which we have used during our analysis of WSN operating systems are also described. These tools are: Surge [9], Listen [10], Oscilloscope [10], Trawler [12], Xnp & Deluge [11] and BTnodes tools [44].

5.2.1. MANNA

MANNA [38] is a policy-based sensor network management system, in which the management is carried out according to pre-defined policies. These policies are executed by the central nodes in case of centralized management. In de-centralized management, delegated agents or cluster heads are responsible for implementing the management policies. MANNA is a collection of recommendations to have an integrated management framework. MANNA manages nodes according to different models such as topology management, residual energy map, sensing coverage area map, communication coverage area map and audit map. Management information is aggregated to central points and then mapped to the pre-defined model maps that initially set by the technicians or set by the WSN. According to the mapping results, a suitable response could be given to

achieve the management. Practically, MANNA has only been set as a management framework for fault management in [43]. Although, MANNA draws the optimal management architecture, it provides no clear information about the feasibility of applying this framework in real WSNs. It proposes a management protocol profile which might be adequate for each WSN applications. However, many missing factors have not been clarified. For example, the energy efficiency, robustness, adaptability, memory efficiency and scalability are still not addressed in this study.

5.2.2. Nucleus (SNMS):

Nucleus is also named Sensor Network Management System (SNMS) [14]. It is a management system that provides several kinds of testing possibilities for managing nodes in the field, such as testing network performance, testing of network connectivity and verifying nodes' presence.

Figure 5.1 shows the essential components of Nucleus and its supporting features. Nucleus supports a query system to get information from log files in the memory. This query system is not based on querying by name. Instead, it assigns, at compile time, an integer key to each name of the attributes.

Furthermore, In contrast to the old fashion in retrieving attributes, which is sending periodic debug messages defined at compile time. Nucleus retrieves the attributes, which the user is interested in observing or monitoring, on request from the running nodes.

In addition, it can be signaled by predefined events occurring on running nodes. Nucleus supports sending the node into sleep mode remotely and waking it again. It can also check some physical parameters of the nodes such as the battery voltage.

Instead of using the application network stack, Nucleus uses a light weight networking stack. This networking stack is based on two main components which are:

- Collection layer
- Dissemination layer

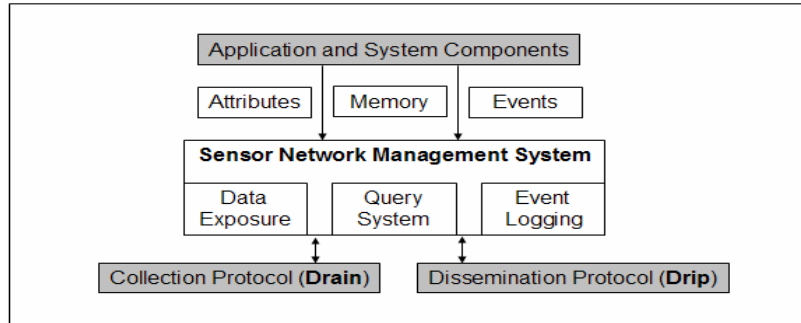


Figure 5.1 Nucleus structure

The dissemination layer, which is called “Drip” [61], is responsible for routing the information to the nodes. This layer also sends queries and commands to the nodes. In addition, by sending the messages multiple times, this layer ensures that the messages are received by the recipients. The dissemination layer has mechanisms to direct commands or messages to a particular set of nodes or to a single node. In case of a single node, the messages are actually sent to every single node; however, they are discarded on all the nodes other than the target one.

The other layer in Nucleus is the collection layer, also called “Drain” [61]. It collects the required messages and directs them either to a single root or several roots. Drain uses the signal strength RSSI to construct the collection tree. The rout update messages are sent by the roots whenever the management of nodes is needed.

In order to ensure the functionality of the management tool, when the original application fails, the Nucleus tool uses its own network interface (“Drain” and “Drip”) alongside the application network stack. This can be considered as a weak point of Nucleus, because the collection and disseminating layers (Drip and Drain) represent redundant routing of the traffic as well as they consume more memory.

Nucleus, as mentioned earlier, encodes attributes name with numbers in order to increase the efficiency of querying because it is easier to query by numbers instead of long strings. This implies that, the correct decoding information is always required when the query is performed.

5.2.3. Surge Network Viewer:

Surge [9], is a java application that runs on the PC which represents the client in wireless sensor networks. It comes normally with TinyOS tools distribution. It provides many

management services to the developer or the user of the wireless sensor network. Some of these services are following:

- It displays the nodes, the links between these nodes and the rate at which these nodes are running, in the form of a graph.
- It provides control on some of the parameters in the application layer. For example, user can set the sampling rates of the nodes.
- It analyses the network log files and shows statistics accordingly such as, the yield (percentage of received packets), link quality yield (link yield to parent) and predicted yield.
- It keeps track of the changes in topology and shows them graphically.

Although, Surge is a handy research management application but it is still limited to only observe the network layer and to interact remotely with only one parameter in the application layer (sampling rate).

5.2.4. Listen Tool:

Listen [10] is a java shell-based tool which is used to display packets on the screen. This is a completely passive tool because through it the user can not interact with the nodes; it is used just for receiving the packets through the PC port and displaying them for monitoring purposes.

5.2.5. Oscilloscope:

Oscilloscope [10] is used to graphically display the sensor readings coming from the nodes in the network. It simply represents the number of packets on the X axis and the reading values on the Y axis. Oscilloscope, shown in Figure 5.2, is also a kind of passive tool, because it does not support any kind of control on the sensor nodes which are being monitored.

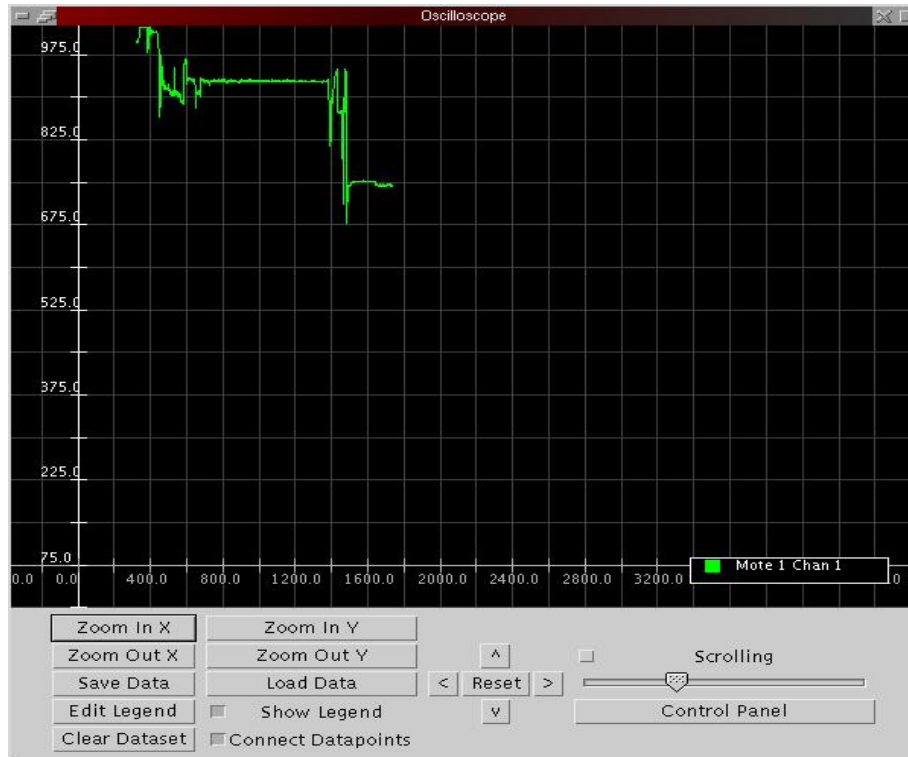


Figure 5.2 Oscilloscope

5.2.6. Trawler:

Trawler [12] is practically a combination of Surge and oscilloscope. It is designed by MoteIV Corporation [13]. This tool has three main tabs; the first one displays the topology of the network and the link quality between the nodes. It also shows the sent, received and dropped packets form every individual node. The other tab shows the sensor reading in the form of graphs for all nodes existing in the network topology. Furthermore, it provides options to zoom in or out these graphs. The third tab shows the link quality of the nodes in the network in graphs form. Figure 5.3 shows the graphical interface of Trawler tool.

Trawler provides options for logging the sensor readings into files. Although Trawler includes a multiple features to assist with data collection, but it is also kind of just passive monitoring of the nodes in WSNs.

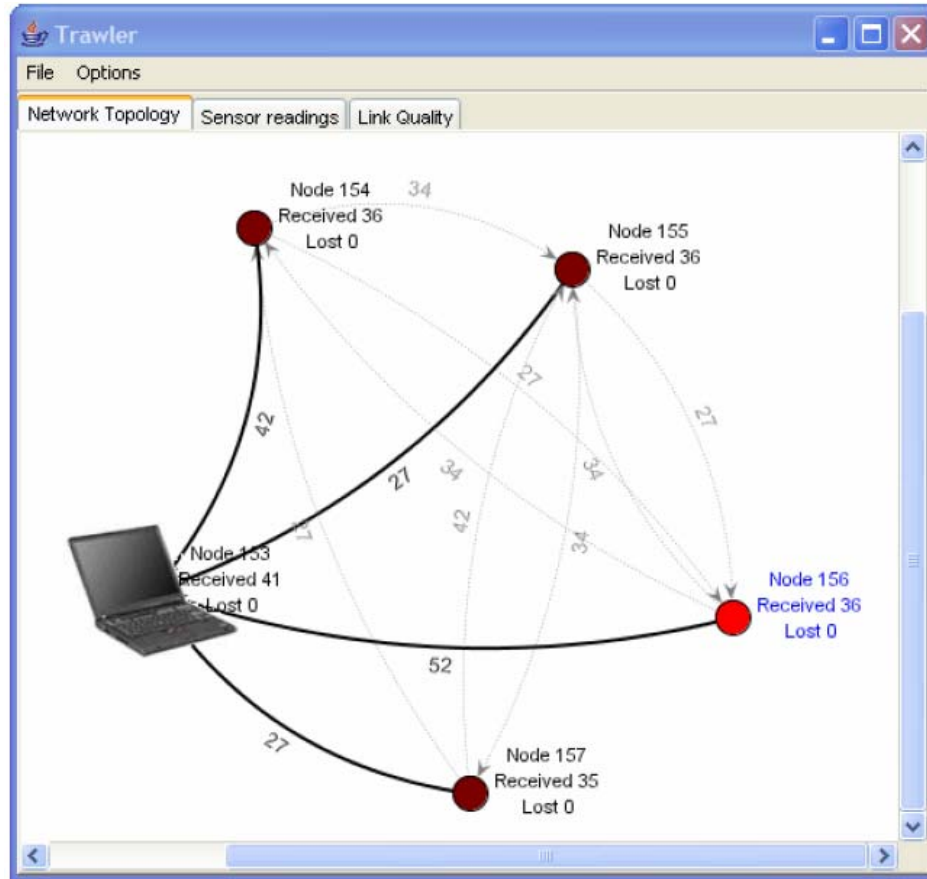


Figure 5.3 Trawler

5.2.7. Xnp & Deluge:

These tools support the remote reconfigurations of the sensor nodes. In both tools, the complete binary image of a new application code is transferred wirelessly to all the nodes and then the application code is activated remotely from the base station.

Xnp [10] was first designed by Crossbow Incorporation [9] and then was modified by Berkeley University. It consists of three parts:

1. The network programming modules
2. Boot loader on the nodes
3. The host tool on the PC side

The programmer uses Xnp host tool to send the program code to the nodes. After every node receives the program code, it checks the sequence numbers of the program lines to verify the validity of the code. If Xnp module figures out some missing lines, it queries to send, once again, the missing code lines. After receiving the complete code, the network

re-programming module on the node, activates the boot loader that starts copying the code to the flash memory. To run the code, the boot loader reboots the system by turning on the watchdog timer, which is responsible for resetting the system. Xnp broadcasts the complete images in a single hop only and does not support the multi-hop.

Deluge [11] is another tool used for on-the-fly reconfiguration of sensor nodes. It uses a data dissemination protocol. Deluge addresses many of the drawbacks of Xnp. It supports multi-hop networks through epidemic dissemination and redundant data integrity checks. Deluge has an improved boot loader called (TOSBoot).

By using Deluge, the user can have multiple program images simultaneously in the nodes. Moreover, it is possible to switch between these images. While using Deluge, a piece of code, called Golden Image, is installed permanently in the nodes' external flash. This piece of code is used for recovery and it is also responsible for the switching among the different code images.

If the user wants to modify some parameters using Xnp or Deluge, the user has to recompile the original system with the modified parameters on the PC and then re-program the nodes wirelessly. Still it is not the optimal and the fastest configuration management to observe the effects of modifications on the nodes as the user has to re-compile and re-program for each and every modification. This would effectively increase the energy consumption because of exchanged code images. Another drawback of these tools is that the different images of the applications have the repeated components that can be shared among several applications. This will decrease the size of the available memory. For example, all programs have some operating system components such as the CPU-Scheduler component. It would be beneficial if they share the same components instead of having multiple copies of these components in every application code image.

5.2.8. BTnode Tools

Most of the above mentioned tools are only designed for MICA sensor node series and tested on it. The Distributed System Group [76] at ETH Zurich has designed its own tools for its BTnode sensor nodes series. Two main tools are implemented by M. Ringwald et al. called SNIF [44]. The other tool for BTnodes is described in "Interactive In-Field Inspection of WSNs" [45].

SNIF: SNIF management tool has four major components:

- Description language for the packets
- A sniffer configured to work with arbitrary radio configurations
- Data stream framework to analyze the packets after decoding
- GUI, graphical user interface to visualize the topology of the network

SNIF, shown in Figure 5.4, assumes that there exists another additional ad-hoc network that sniffs the inspected wireless sensor network. Since not all the messages from all the nodes might be overheard by the sniffer network, this method will never give the full status of the network. Another drawback of this tool is that it is not based on special management packets; rather it uses the original traffic received from each node to decide on the status of a particular node. But this will not cover the status of the nodes that stop sending their messages right from the beginning of running the application.

The other tool, which is referenced with [45], supports three main services which are: ‘Re-tasking’, ‘Attribute query’ and ‘Topology visualization’. Re-tasking, in this tool, provides means to program the nodes wirelessly through using external memory of type SRAM. After rebooting the system, the image stored in the SRAM will be transferred to the flash ROM. Vital attributes, in this tool, can be queried frequently. Finally, visualizing the network topology in the tool is not provided by the tool itself, but it is taken from the application running on the nodes.

All these described tools do not provide overall control and management of the system components such as the network layer, operating system parameters and application settings in real sensor nodes. This lack is addressed by our designed tool that provides adequate management of the nodes in the field. Additionally, it introduces a new management framework which mainly increases the interactivity with sensor nodes, and provides monitoring of the network as a whole as well as for every individual node. Moreover, this framework aims at providing more setting choices (performance and configuration management) and more elaborative graphical interfaces.

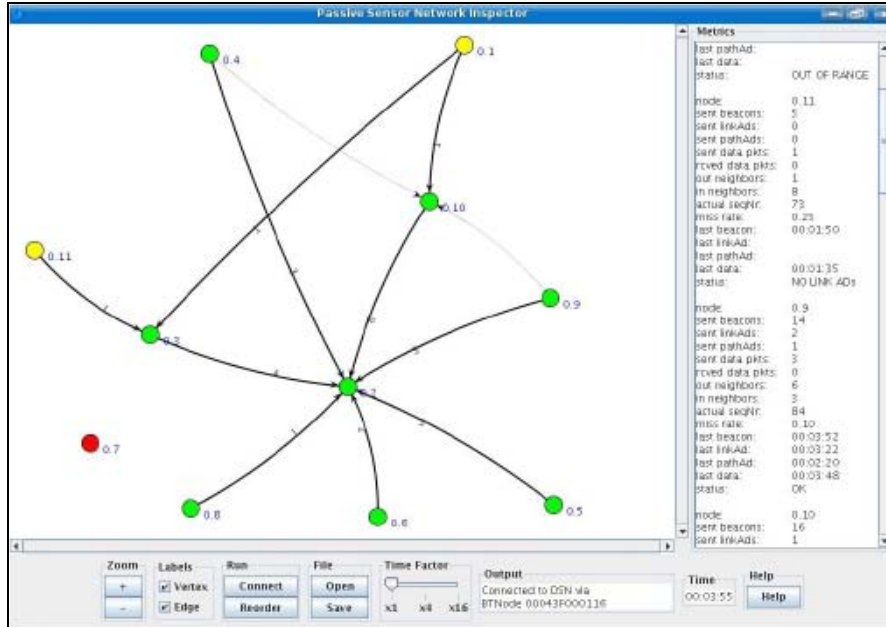


Figure 5.4 SNIF, Sensor network inspection framework

Chapter 6

Requirements, Architecture and Features of ITSN

In this chapter, requirements of our designed tool are discussed. Then, its architecture on nodes side and the computer side is elaborated, since the tool is based on two main collaborative parts, the first exists on the individual nodes and the other exists on the computing device. Finally, the features of generality, management efficiency, fault tolerance supported by this tool are discussed.

6.1. Design Requirement

There are several prerequisites needed for our proposed application to function. The proposed application does not demand any additional hardware equipments; rather it requires software modules such as information dissemination module, information collection module and, dispatching layer between nodes and the PC. These requirements are discussed in the following sections.

6.1.1. Information Dissemination

In WSN management, the data dissemination is the process of routing the management queries and commands to a specific node, to multiple nodes or to all the nodes in the network. Data dissemination is a wide research area in ad-hoc networks. Mainly, the dissemination protocols can be periodic, on demand or event based. In the following, some dissemination protocols are first described. Then, the dissemination scheme adopted in our tool is further elaborated.

- **Flooding:** In this method, every node re-broadcasts each packet that it receives. This broadcast is repeated until the flooded packets reach the network boundaries.

Although, this method has many drawbacks such as implosion, overlapping, and high energy consumption. Implosion means that a node receives duplicated messages. Flooding is used in many WSN applications. The main advantage of this method is its simplicity. In directed diffusion which is one of the key schemes in WSNs [46], flooding is used to propagate the interest of tracking objects to other nodes.

- **Gossiping protocols (sometimes are called epidemic protocols):** These protocols follow an approach similar to flooding. However, in gossiping every node sends the received packet to one neighboring node in a random manner. Gossiping is not a reliable communication method. However, it is still suitable for many WSNs applications due to its lower energy consumption and lower channel bandwidth. An example of gossiping protocol for WSN can be found in “Adaptive Probabilistic Epidemic Protocol for Wireless Sensor Networks in an Urban Environment” [48].
- **Ideal routing:** This approach includes both the uni- and multicast routing. In ideal routing, a global knowledge of the network conditions is required by the nodes to achieve the best performance as compared to the other data dissemination schemes. However, the awareness of the network conditions demands a huge amount of the node resources such as memory, channel bandwidth, energy, computation etc. Many derivations of ideal routing protocols are suggested for WSNs such as “Adaptive Demand-Driven Multicast Routing in Multi-Hop Wireless Ad Hoc Networks” [49].

In our scheme, the simple flooding is used due to its simplicity. However, other schemes can also be used. The deciding criterion for choosing a dissemination scheme is based on the user preferences and the application requirements. For example, very large scale networks perform well with gossiping protocols due to its energy and bandwidth efficiency, while in moderately dense networks, the flooding schemes perform well due to its simplicity. In small scale networks, where the bandwidth can accommodate control traffic, the ideal routing would be a good choice.

6.1.2. Information Collection

The information collection, which is the opposite process of the data dissemination, is the process of fetching the data from the sensor nodes to the sinks or to the central nodes. In Information Collection, spanning trees are first built and rooted to the central node. This scheme has been followed in “A Tiny Aggregation Service for Ad-Hoc Sensor Networks” [50] and “Synopsis diffusion for robust aggregation in sensor networks” [51]. In our management application, the Collection Tree Protocol (CTP) [52] is adopted as an information collection scheme. This protocol uses the link quality indicator as a metric to select the best parent to establish a routing tree. Every node in the topology maintains its routing table where it saves the possible available parents. In addition, many other fields are also stored for each parent such as node ID, number of hops to reach the base node, the estimation of the link quality etc. CTP should address many inconsistency issues while collecting the data such as packet duplication and loops. Figure 6.1 describes an example of a collection tree. This collection tree collects information from the WSN and forwards it to the base station from where it can be observed by the users.

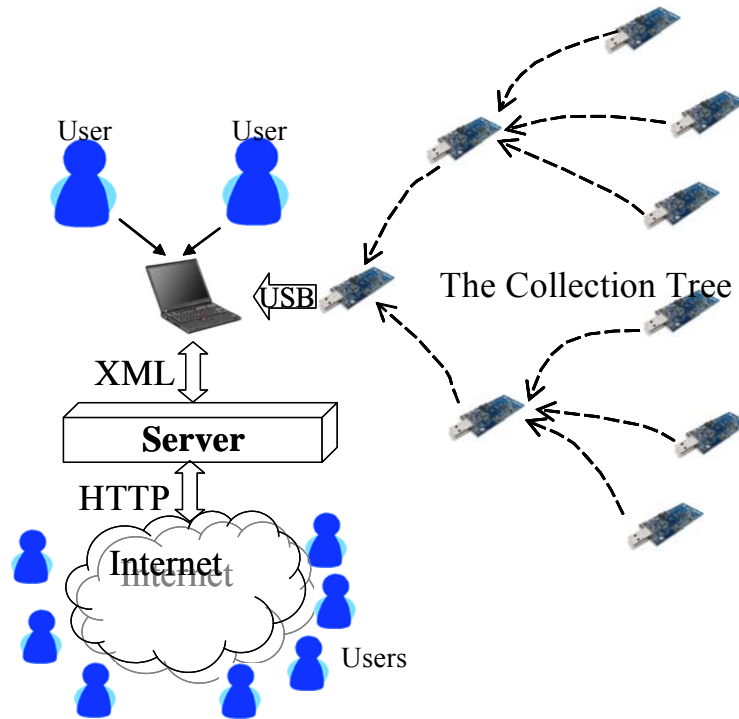


Figure 6.1 A representation of a collection tree

6.1.3. On-Client Dispatching Interface Layer

As mentioned earlier, the nodes represent the server because they are the actual place where the data is originated and processed, while the facilities which are on the PC side are the clients from which the requests are being sent.

On the client, there is a need for an interface that can provide a low level communication layer between the PC and the node. There are many reasons for having such an interface. For example:

1. It provides a dispatching layer between the nodes to the PC's programs communicating with the nodes and vice versa. Especially, it provides an interface to access the nodes remotely from the services on the PC.
2. It establishes and synchronizes the connection between the nodes and the PC.
3. It resolves the encoding and framing heterogeneity between the sensor nodes and the PC communication stack.
4. It resolves the heterogeneity caused by the wide variation of hardware communication possibilities such as Bluetooth, serial ports, Ethernet etc.

In our tool, earlier we have designed our interfacing program to establish the communication between the base station and the PC. However; later the "SerialForwarder" interfacing layer has been adopted. The reason was that it can support different types of sensor nodes. "SerialForwarder" is used to connect nodes through PC interfaces such as serial port, Ethernet, etc. It uses the „Packetizer" Protocol to transmit data, which is inspired by the Point-to-Point Protocol (PPP) reliable transmission protocol in (RFC 1663) [79].

6.2. Architecture of the designed Tool

The designed framework [4], shown in Figure 6.2, has two collaborative parts, one is running on the PC (the client) and the other is running on each node (the server). These two parts on the node side and on the PC side communicate and cooperate to achieve the management, visualization and remote controlling tasks. In this section, the architecture of the management framework is explored in detail. Firstly, part which is on the PC side is discussed. Next, the other part which runs on every individual node is discussed. Then, the approach, how the two tool parts collaborate to process the exchanged packets, is explained.

6.2.1. PC-Side Components

The part of the management framework on the PC consists of several components. As depicted in Figure 6.2, these components are associated with corresponding layers that exist in every node in the “Thin Layer”. In the same figure, the “Thin Layer” represents the tool ends on the sensor nodes. It is responsible for controlling the application and the system on the sensor node.

As this tool resides on a strong computing machine (such as PC), having huge resources as compared to the sensor nodes, it can provide more functionalities. For example, it provides a repository of data coming from nodes, visualizes the information in graphical form, and conducts analysis of the messages exchanged with the node in WSN.

This part has several independent components to provide multiple and modular features. All these components are categorized into two main types, which are:

1. PC Sensing components: These are responsible for dealing with the sensing data on the PC. They receive the sensing data, save it and analyze it. They keep track of the packets, received from every node, and of the sensed parameters such as temperature, humidity etc.

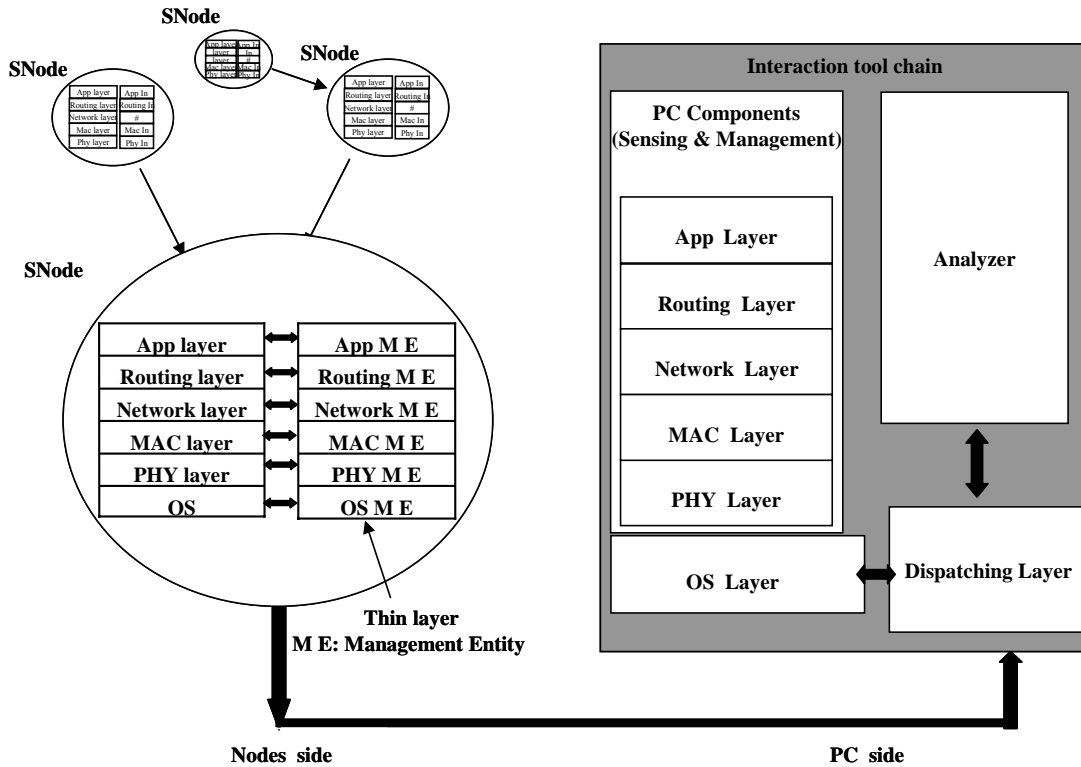


Figure 6.2 General architecture of the management Tool

2. PC Management components: These components deal with the management data on the PC. They remotely manage the system parameters inside every node by controlling the corresponding layers via the management entities in the “Thin Layer”. For example, the MAC layer component of the tool on the PC controls and manages the MAC layer of the sensor node via the MAC management entity (MAC M E in Figure 6.2) that resides on the node. The MAC layer component also analyses the performance, and displays the results in graphs forms. As another example of such components are network layer management components, physical layer management components, operating system management components.

6.2.2. Nodes-Side Components

The components of the tool on the node side (management entities) should be of small-size code and utilize a little amount of the processing power. These components compose the “Thin layer” in the node, Figure 6.2. Because of the rapid developments in sensor

nodes, the “Thin layer” should not significantly decrease the node efficiency. It is observed that the memory size in sensor nodes is getting larger and the processor speed with same amount of energy consumption is getting faster day by day. This implies that this layer can be embedded easily on the node side without having considerable implication on the efficiency of the sensor node. This “Thin layer” can be added as a module to the system and is optionally integrated to it on the node. Furthermore, it can be remotely disabled or enabled according to the user requirements.

In the first outline of the interaction tool, we have assumed that it provides a full control and management of all the nodes in WSN. Moreover, it controls the functionality inside each node separately. The “Thin layer” of the tool on node side has already many ports shown in Figure 6.3, which are the parameterized interfaces in the nesC language. In this figure, the overall structure of the components inside the managed node is shown. The system components are connected with the corresponding management components in the thin layer through the ports.

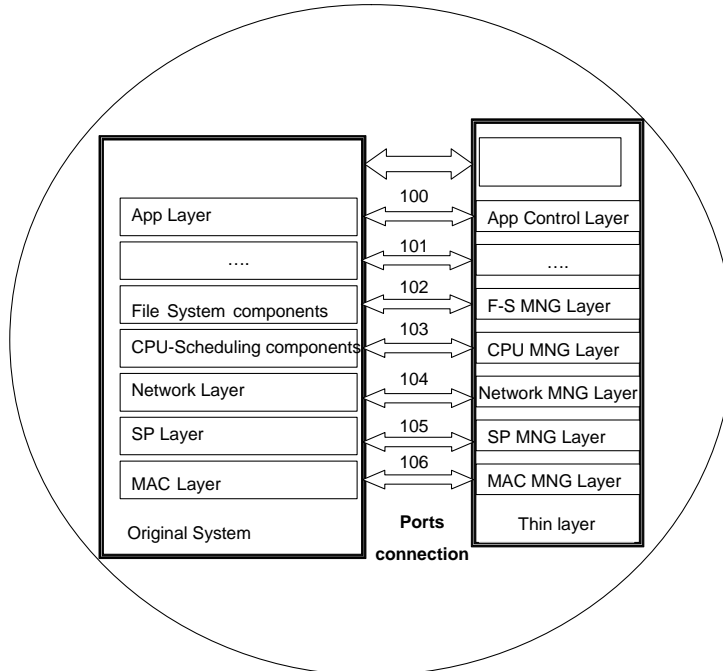


Figure 6.3 Architecture of the sensor node including the management components

6.2.3. Communication between Tools Parts (on PC & Sensor Nodes)

Any additional traffic in WSN should be dealt with very carefully due to the narrow bandwidth. Although, the communication between the tool's ends (the management entities on nodes and the tool on the PC) adds additional control traffic and consume more energy, but obtaining and managing the parameters is necessary to take important decisions, which can help in improving the overall performance of network.

The communication of the framework has two modes, as seen in Figure 6.4:

- Passive mode: in which the tool is only a monitoring and visualization tool of the nodes without having any kind of interaction with the nodes. This visualization is based on the sensing data packets forwarded to the base station node.
- Active mode in which the tool receives packets and responds accordingly to a particular node or to a group of nodes in the network. In this mode, the tool also can set or change the settings of the individual nodes.

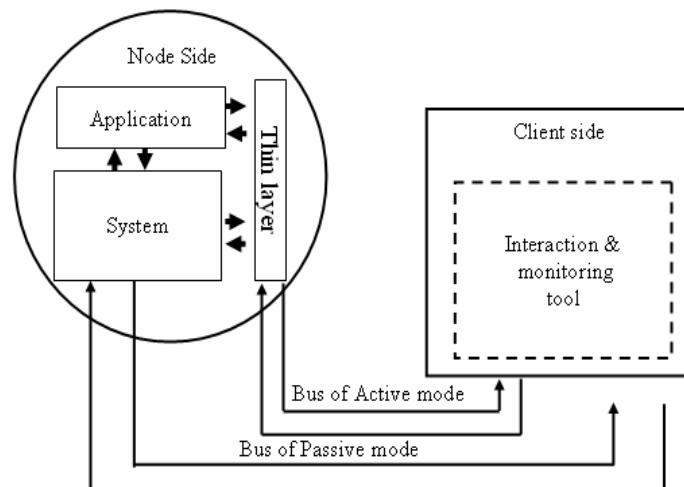
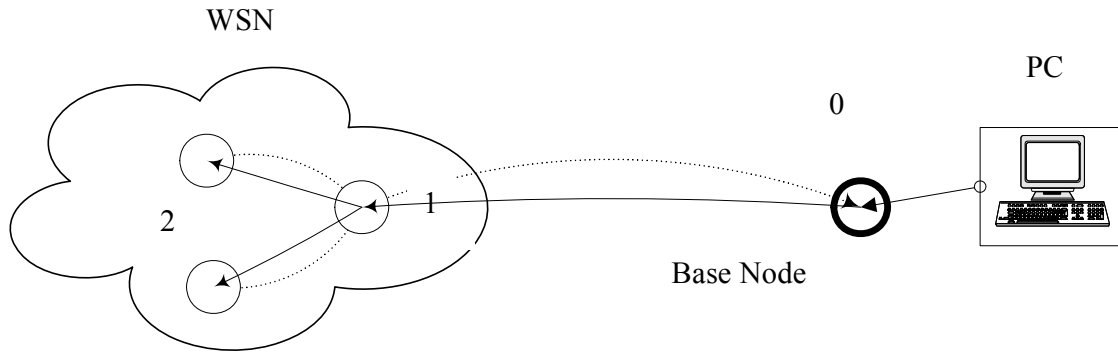


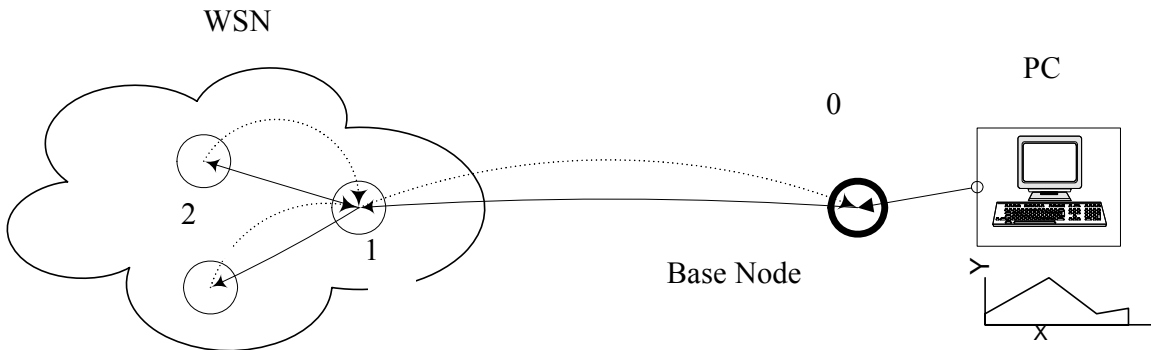
Figure 6.4 Communication modes in the management tool

The communication between the tool ends on PC and on the sensor nodes, depicted in Figure 6.5, takes place as follows. After the deployment of the nodes, the aggregation protocol establishes a collective tree. The aggregation protocol uses hierarchical routing scheme based on the link quality estimation. It first establishes gradients, which are set up upon the announcement of the base station that it is in depth zero. Then, its immediate sub-nodes will be in the depth one in the routing and so on. After establishing the routing tree, every node starts sending the sample packets to its parent in the tree. Eventually, all packets are forwarded to the base station node and then to the PC, where the information is analyzed and displayed to the user. To send a packet reversely from the base station to the nodes, flooding is used in our scheme.

Let us take an example to explain how to query the nodes and how the response is sent back to the base station. In this example, few parameters in the MAC layer are queried. First, the whole network is flooded with the query or the command. The nodes are signaled from the base node which is initially signaled by tool on the PC side to broadcast the query. After each node receives the query or the command on the corresponding MAC port via the management entities, it responds to it by answering the query. It also responds by re-broadcasting the query to all of its neighboring nodes. This messaging process is very simple because of its broadcast nature. It also takes a short time comparing to uni-cast communication which needs to forward packets through already specific reserved paths. Finally, each queried MAC layer starts issuing the particular query answer. The query response will be returned to the base node in a uni-cast fashion, so that every node sends to its parent node. These messages are forwarded via the intermediate nodes until they reach the base station and then they are stored in a data repository for later analyses. These messages are also used to generate different informative graphs inside the framework (tool).



Step 1: Base node is signaled from the PC and then the Base node floods the nodes in WSN to initiate the routing tree (flood messages are continuous arrow)
 Step 2: After topology has been established (dotted lines), the base station will be in depth (0), next routing level will be depth (1), the following will be (2).



Step 3: Command can be sent to nodes using flooding (flood messages are continuous arrow)
 Step 4: Management packets are sent back to the base station in Uni-cast fashion (dotted arrows)

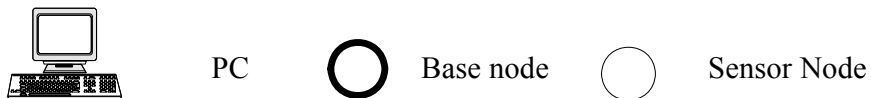


Figure 6.5 Communication between the tool's parts

6.3. The Main Features of the Designed Tool

The management tool has many features that make it a strong framework in managing and controlling sensor nodes in WSNs. In this section, we have classified the generic management solutions of WSNs. Next, the generic nature and the efficiency of the tool are discussed. Then, the supported fault tolerance model is described. Finally, the plug-in feature provided by the tool is elaborated.

6.3.1. Generic Management Applications in WSNs

Ignoring the generality of WSN applications during the design phase limits their benefits. Furthermore, this would restrict the possibilities of future extension and adaptation. Several methods to enhance the generic nature of applications in WSNs are explained later on. We have further evaluated the suitability of these methods in centralized and decentralized management scenarios [80].

Providing generic platform- and hardware-independent applications for WSNs is very important in order to ease the use of the big diversity of WSN applications. Moreover, in WSNs, finding this generic management architecture that can be reusable for multiple sensor node platforms is a challenge being posed and emphasized by the WSNs community for long time.

Initially, this question should be mapped into the current traditional network management problems, because the learned lessons of the traditional networks in this regard should be exploited. Therefore, we have classified the similarities and distinctions between the traditional and wireless sensor networks from the management perspective.

In traditional networks, the generic nature of the application provides several important benefits such as:

- **Compatibility:** this enables the users to benefit from different components for the same platforms.
- **Interchangeability** of the same components among different platforms while providing the same functionality.
- **Commonality:** it is to have the similar way of dealing with the components.

One of the main challenges of WSN applications is the specific nature. Therefore, it is very difficult to find generic solutions. This specific nature is due to many reasons which are as follows:

- The limited resources in nodes such as small size, small memory, weak processor, narrow bandwidth ... etc.
- The wide range of applications in which we can use WSNs: In WSNs, there are many categories of applications such as, environmental tracking applications, medical and industrial applications, home automation application, surveillance systems ... etc (See Appendix A).
- The diversity in hardware component choices: many hardware technologies are in use in WSNs applications such as: several types of microcontrollers, several types of transceivers and many other diverse hardware elements. This diversity requires a specific way of designing and setting.
- Over the last few years of WSNs evolvement, a lot of software solutions have been introduced which has increased the software diversity, and hence complicated finding general software solutions.

In the following, we provide a survey of the methods employed to enhance the generality of the management in WSNs.

Middleware:

In WSNs, Middleware can be used to overcome and address the limitations of the application-specific nature. It also supports the commonness of the systems, deployment, development and maintenance. Many middleware solutions for WSNs have been proposed. Hadim et al.[59] have covered in their survey a large number of the WSN middleware proposals such as: Mate [68], TinyDB [69], SINA [70], Agilla [71] and many others. All these middleware systems support three main objectives:

1. Ease the use of the applications
2. Resolve the heterogeneity between different platforms
3. Openness

Dynamic and Mobile Agents:

Mobile agents, in this context, are small pieces of code which can be exchanged between the nodes in order to provide the generic nature to the application. Each manufacturer of a sensor node type provides an agent which comprises the node specifications. These specifications are later used to enhance the compatibility and to resolve the heterogeneity with other node types. In this paradigm, the sensor node can be seen as any computer device which has multiple software drivers in order to be compatible with other hardware platforms.

Semantics Methods:

This method is based on an agreement of the functional meanings of the data structures and the functionality of the WSN specific components. Here, the exchanged management messages among heterogeneous application can semantically be interpreted in order to produce a general messaging structure. This general message format can be later used to provide general management. This method is used in Abstract Syntax Notation one (ASN.1) which is used in Simple Network Management Protocol (SNMP) in TCP/IP model.

Standardization

Standardization is a way to have compatibility, interchangeability or commonality among multiple systems based on different technical and operational fields. In WSNs field, there are few standards (technical agreements) that have been adopted such as ISO-18000-7 [54], 6lowpan [55], WirelessHART [56], ZigBee [57] and Wibree [58]. These standards are not developed explicitly for WSNs; rather they are mainly proposed for supporting general low power and low rate networks, and hence they can be adopted by WNS technology.

SP [30] (Sensor net Protocol) is, so far, the only standard-alike that has been specially proposed for WSNs. SP is further explained in the following subsection.

6.3.2. The Generic Management Nature in the Management Framework

From the very start, we have designed our management framework in such a way that it should provide certain degree of generality for distinct wireless sensor nodes. On the first design stages, we have classified the node components into two categories: First, node-dependent management components. These components can not be shared by different sensor node types. Second, node-independent management components which can be shared between different sensor nodes types. This classification helps in finding the components whose heterogeneity should be addressed.

An example of the node-dependent management components is the interfacing layer, which connects between the tool on the PC side and the external nodes and vice versa. In our tool, in order to enhance the generality, we have adopted one of the widely used schemes, although we have designed our own as well. This adopted interface is called “SerialForwarder” [60]. “SerialForwarder” uses the “Packetizer protocol” which sends the packets through different busses between the base node and the PC. The “Packetizer protocol” supports acknowledgements at the link layer.

In order to increase the degree of generality and to deal with arbitrary kind of messages inside the tool, the tool also provides a standard messages interface that is compatible with the messages format generated by the Message Interface Generator (MIG) [53]. MIG is a tool used to generate unified data structure format from multiple languages that corresponds to the original message structure of the nodes. Figure 6.6 shows a representation of how MIG is used to resolve the heterogeneity of different messages structure of multiple kinds of WSNs (SN(1), SN(2), SN (n)). The generated messages are compatible with the tool that we have implemented in JAVA.

Moreover, the tool further enhances the generality by saving the sensing data and the management data in the XML format. The XML files can be used later by the tool to execute actions or to deduce inferences as well as they can be simultaneously accessed by external programs. The user of the tool can select whether to save these XML file to temporary or to permanent files.

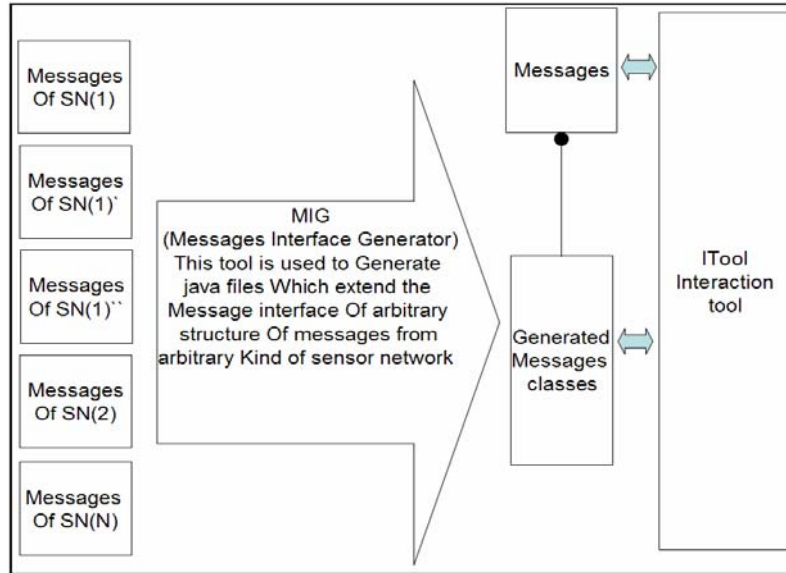


Figure 6.6 Message generation for the tool using MIG

Finally, our tool supports the Sensor net Protocol (SP) [30] functionality which is a step forward towards generalization in WSNs. SP is a unifying abstraction layer that bridges different network protocols with different underlying data link layers and physical layers. SP is not at the network layer but instead it sits between the network and the data-link layer (because in sensor nodes data-processing normally occurs at each hop, not just at the end points). SP can also be used to identify an individual node, a set of nodes, or a communication structure such as a tree. This is an important feature supported by SP, as in most of the WSN applications, addressing a single node is not required; instead addressing a group or a tree of nodes is required.

6.3.3. Management Efficiency of the Framework

Resource efficiency is very critical in WSNs. On the one hand, high number of management parameters would increase the energy consumption. On the other hand, these management parameters are too necessary to take important decisions. These decisions can help in improving the performance of the network. The mechanisms followed to observe and track the system parameters do not significantly affect the data channel throughput or the system workload.

From the packet overhead perspective, a low sampling rate to monitor the nodes should be set so that it should not significantly affect the original traffic. For example, if the user

is interested in monitoring few parameters at the routing layer, the rate of the management packets, which are delivered between the parts of tool-on the PC side and on the node side-should be lower than the original sensing sample rate.

To ensure the scalability, it is assumed that for each certain number of nodes another base node (connected to a PC) should carry out the monitoring and the interaction with its tree nodes. This will reduce the overload on the individual base node connected to the PCs in large scale or dense WSNs.

Flooding is used to disseminate queries to the nodes. Epidemic protocols can also be used, if less energy consumption and low traffic communication are desired.

Uni-casting is used here to retrieve information from nodes because of its reliability. In case of data aggregation applications, data fusion and filtering schemes can be combined to the uni-cast routing to decrease the number of retrieved packets in the collection tree.

6.3.4. Fault Tolerance Model of the Framework

As mentioned in chapter 4, fault tolerance is one of the five dimensions of network management. The framework can be integrated with different fault-tolerant schemes. To present this feature, we have derived a simplified version of a general fault tolerance model provided in “Tolerating Failures of Continuous-Valued Sensors” [72]. This derived version is very close to the fault tolerance model proposed in “A Geometric-Based Approach to Fault-Tolerance in Distributed Detection Using Wireless Sensor Network” [72], [73]. The derived model is simply based on analyzing the geometric topology of the nodes and also based on the overlapping of the output values of the sensor nodes. In other words, if part of the nodes is faulty, the combined output with others is more reliable. The fault-tolerant model assumes a continuous event region which means that the events of the measured parameters are not centralized in specific points; rather, they are distributed evenly over the field. Moreover, this model is not based on the probability schemes.

The first step in this fault-tolerant model is specifying the sensing map. The sensing map is based on the nature of the sensed parameter and on the environment specifications. For example, if some sensor nodes are deployed in forest, part of the nodes will be deployed in the shadow or in dark places while their neighbors will be in sunny areas. This leads

that sensor reading values of the illumination vary highly. In contrast, the temperature sensing map would not have that large variation of the sensor reading values. Table 6.1 contains temperature and illumination sensor reading values of eight sensors of type (Tmote). Part of them is deployed in the shadow, while the others are exposed directly to the sun, which would be realistic in a forest deployment. In this particular case, (1, 2, 3, 4) are deployed in the shadow and (5, 6, 7, 8) are exposed to the sun.

Figure 6.7 represents the illumination and the temperature sensing maps of the sensor reading values which are in the Table 6.1. Group (A) represents the nodes 1,2,3,4, and the group (B) represents the nodes 5,6,7,8. These figures demonstrate the distinction between the different sensing maps of the different parameters. In other words, the temperature varies slightly over the field for both groups (A and B), while illumination can have a big deviation over the field. This factor is very essential to be considered in the fault tolerance model of WSN to specify the degree of fault.

Group	Node_ID	Temperature		Illumination	
		Sunny	Shadow	Sunny	Shadow
A	ID_1		24.4 C		77 Lux
	ID_2		25.7 C		42 Lux
	ID_3		25.5 C		52 Lux
	ID_4		26.0 C		38 Lux
B	ID_5	27.0 C		900 Lux	
	ID_6	26.0 C		950 Lux	
	ID_7	28.5 C		873 Lux	
	ID_8	26.5 C		843 Lux	

Table 6.1 Sensing values of sensor nodes deployed in a forest field

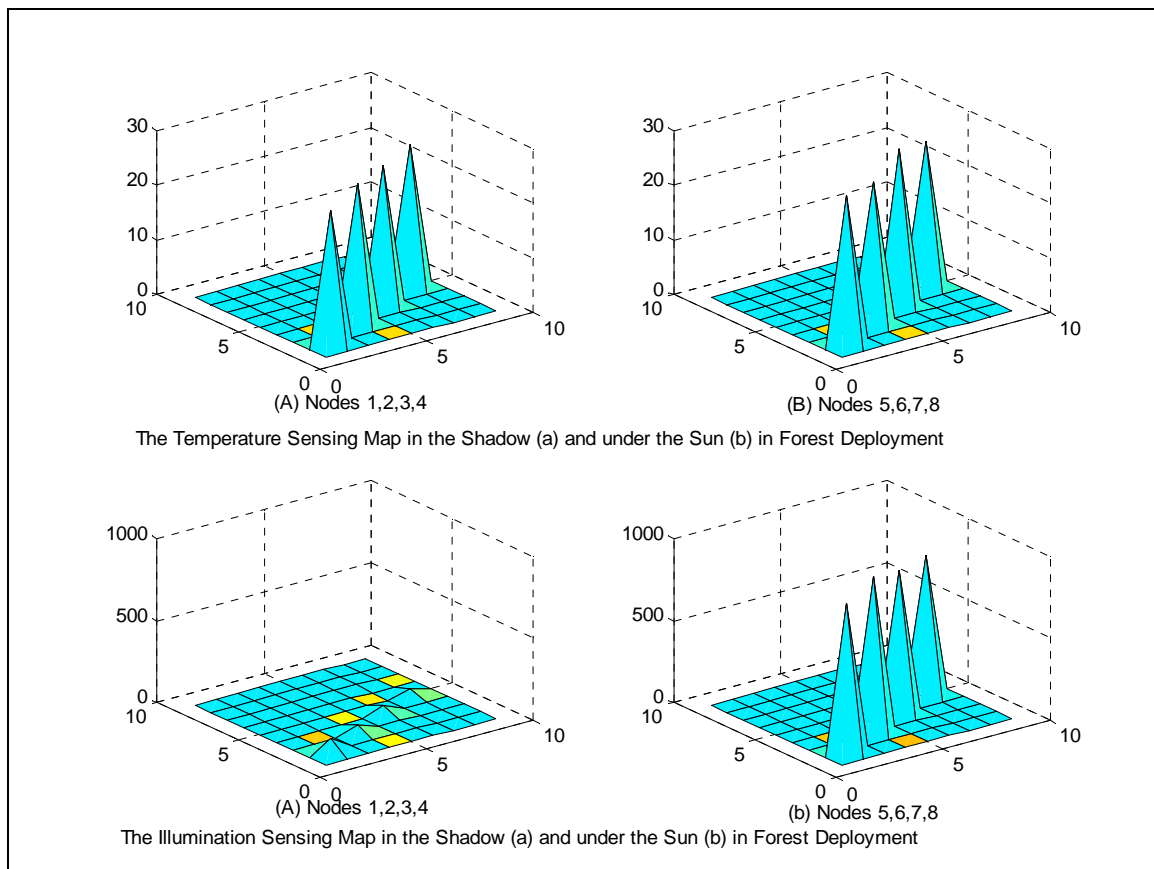


Figure 6.7 Sensing maps of temperature and illumination of a forest deployment

The deviation degree of the sensing map should initially be decided by the user to be later considered by the fault tolerance scheme. In the management framework, the user specifies via a dialog box the highest deviation of the sensed parameter. This deviation is dependent on the environment and on the nature of the parameter. Based on the specified sensing deviation, the framework correctly references the fault degree in the continuous event region as follows. Every node announces its Geo-position (Longitude and the Latitude) and its sensor reading values. Then, the framework decides which other nodes are in the communication range of the node. Next, the mean value of all sensing values is measured within that range. According to mean value, the tool can estimate the degree of fault of the sensor at that particular region where the node is deployed.

Let us take an example a network of three nodes (N_1 , N_2 , and N_3) that are in each others communication range as shown in Figure 6.8. Their sensor readings are 23, 37, and 30 respectively. The mean value is 30, which means that node N_1 has +7 margins of faults. If the allowed deviation would be ten, it means +7,-7 are still realistic and still in the tolerance limit of the sensing map.

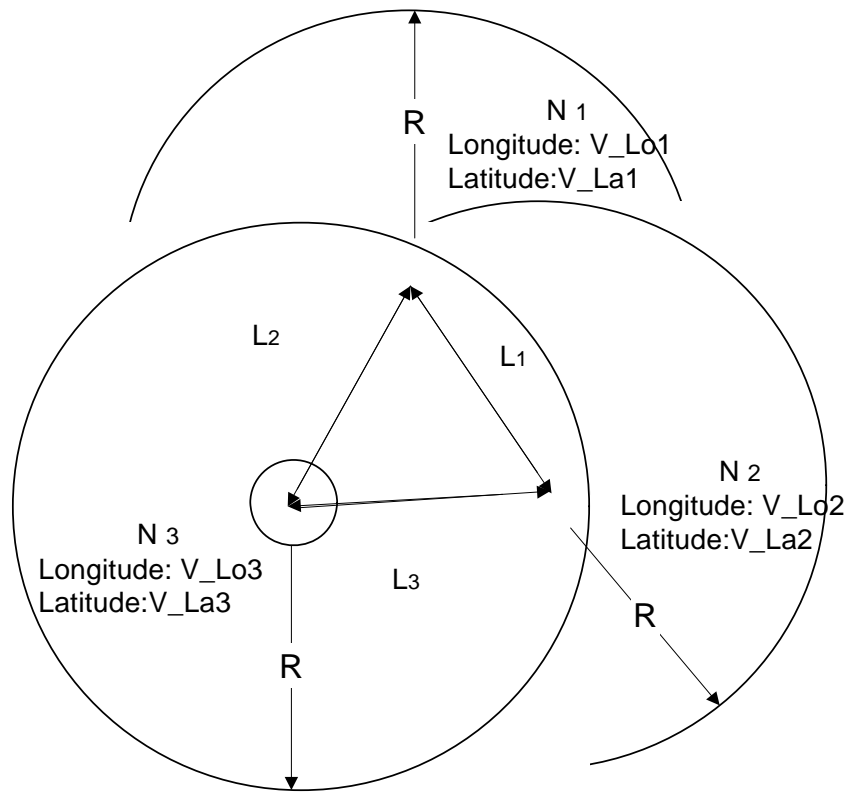


Figure 6.8 Node topology of three nodes are performing the fault tolerance scheme

6.3.5. Plug-ins

An important feature of the management framework is the provision for additional plug-ins. The purpose of these plug-ins is to provide third-party developers an option to extend the tool capabilities. In our tool, a simple public interface is provided to the external users or designers. Through this interface, different tool services can be accessed and modified. Furthermore, the interface provides back signaling with all the potential events that might occur either externally at the nodes or internally inside the tool. As a demonstration of the Plug-ins feature, we have already implemented some plug-ins for the tool. They are explored in the following.

6.3.5.1. MAP Plug-in:

Inside this Plug-in, the user specifies the map parameters (name of the place, address, latitude, longitude, or zip) according to the terrain, in which the user plans to deploy and then monitor the sensor nodes. The Plug-in then makes a request to MAP web services provider. It, then, fetches the corresponding map and displays it in the nodes' panel. This Plug-in provides many options such as: adjusting, zooming in or zooming out the map. Currently, Yahoo Map Web Services are used in this Plug-in.

For positioning or locating the sensor nodes, there are already many proposed schemes such as in [63] and [65]. The user can show nodes in their absolute or relevant positions by using this plug-in if an absolute or relevant positioning algorithm is already running on the nodes.

6.3.5.2. Sensing Information Visualization Plug-in:

When a data packet arrives to the tool, an event will be signaled to the plug-ins which receives this event. This event contains the fields:

- node' ID
- sensing information type
- timestamp of the event
- sensing value

This information will be stored in an XML file.

To display this sensing information, the user has to click on a node in the nodes panel. Then an event will be generated. As a response of that, the XML file will be parsed and a chart representing the sensing information of the chosen node will be generated and displayed on the plug-in panel.

6.3.5.3. Network Information Plug-in:

This plug-in is responsible for obtaining information about the network in the sensor nodes. The user has to enable this Plug-in to monitor the network parameters such as neighbor routing table, signal strength indicator, signal quality and other parameters. Whenever a network information packet is received, an event of type “NetworkEvent” is signaled. This event is sent to the plug-in. The content of the received packets is saved to XML files. These XML files are parsed and then the information is displayed on the plug-in panel.

6.3.5.4. Scheduler Performance Plug-in:

In this Plug-in, the information on CPU scheduler performance of each node is collected and accordingly graphs are displayed. First, an event of type (SchedValueEvent) is signaled to the plug-ins. This event contains:

- node’s ID
- the value of the CPU tasks left on the scheduler
- timestamp of the event

These values with their associated time and IDs will be saved into XML files. Whenever the user clicks on a particular node in the nodes panel, the XML file will be parsed and then a chart will be generated describing how much this particular clicked node is being utilized during time. Figure 6.9 shows a snapshot taken from the tool and it shows the CPU-scheduler utilization of a node. It is shown in the first portion of this graph, the CPU utilization of a node while forwarding packets from other sub-nodes in addition to its own packets while in the later portion it just sending its local packets.

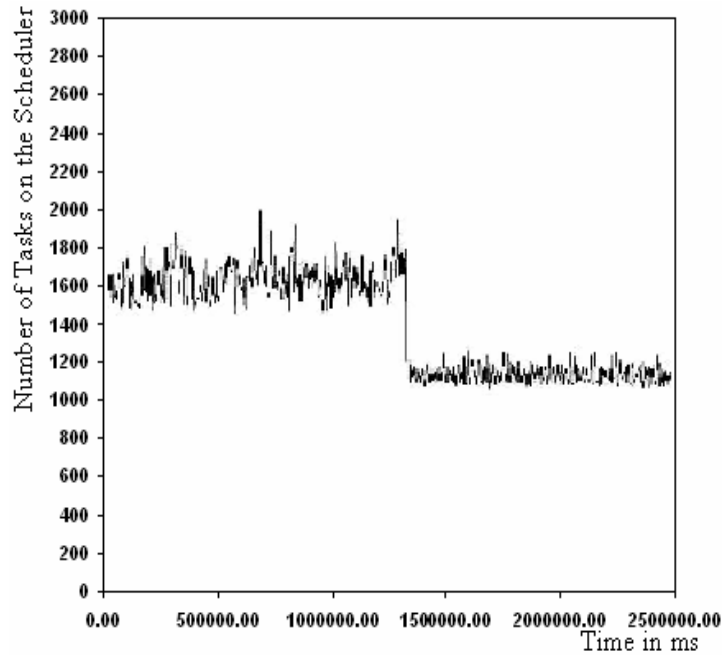


Figure 6.9 CPU utilization of a node

In the following Table 6.2, we summarize the main features supported by the tool.

Feature	Description
Architecture	Hierarchical
Reactivity	Both Proactive and Reactive can be set by the management tool
Generic Nature	Supported via SP, XML, Plug-ins
Robustness	Supported through the fault tolerance scheme
Energy Efficiency	Supported partially by controlling the sample rate of the management packets
Scalability	Supported through adding additional base nodes

Table 6.2, Features supported by the management tool

Chapter 7

Implementation

The implementation of the tool is explored in this chapter. The first part discusses the two main component groups of the framework which include both the components on the PC and the component on every sensor node (management entities). This exposes the implementation of these components and the techniques they perform to achieve the management and to achieve the remote controlling.

The second part of this chapter discusses the plug-ins implementation. It explains the advantages of this feature. It also explains the Application Programming Interface (API) of the Plug-in for adding additional plug-ins.

The tool module on the PC is based on JAVA technology, while nesC is the programming language used for the modules in the sensor nodes. nesC is used for the implementation of the operating system “TinyOS” that runs on the sensor nodes in our used development kit.

7.1. Main Classes of the PC-Side Components

Figure 7.1 shows a UML diagram of the main classes of the tool. In this figure, the class “InterTool” is the main class which launches the functionality of the tool utilities classes such as the “Nodes”, “ToolFrame”, “Eventholder” and “MessageInterface”. The “Nodes” class holds the information of the nodes found in the topology and their status. “ToolFrame” is responsible for rendering the visualization. “MessageInterface” is responsible for communicating with the nodes via the base node. “Eventholder” is responsible for saving all the internal and external events and then signaling them to the other listening classes.

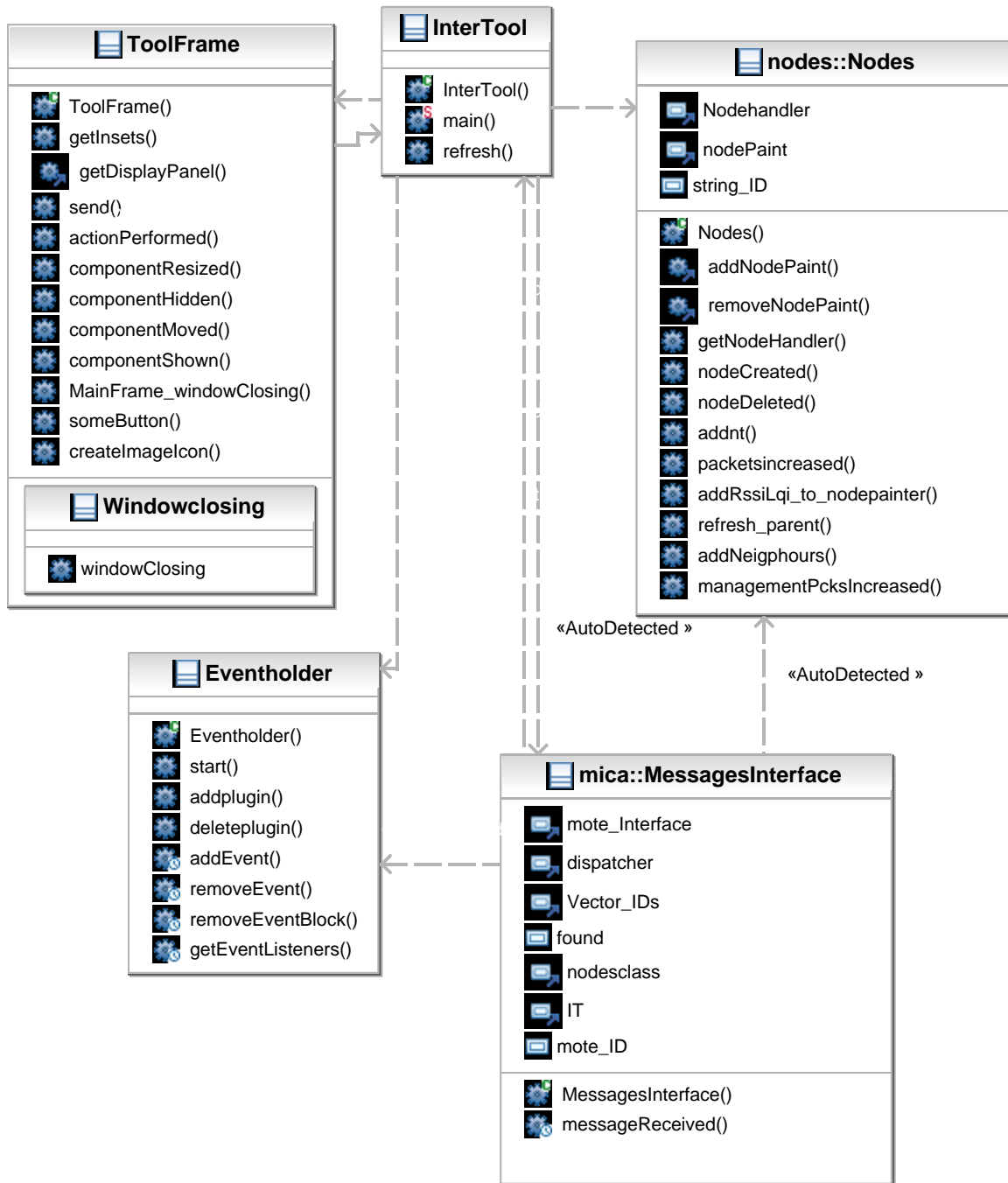


Figure 7.1 UML diagram of the main classes inside the management tool

The tool combines several groups of components on the PC-side. Each group of these components implements and performs certain functionality and logic. These groups of components are classified into the following main categories:

- **Messaging interface components**

They provide a standard bi-directional interface which dispatches messages to the other tool components, after omitting the link layer headings. The received messages are switched to the corresponding components where they will be further processed. For example, it forwards messages from the nodes headings towards the routing layer components, link layer components .etc. This interface also forwards the messages sent to the nodes from the components that are on the PC-side.

- **Visualization components**

They receive the messages from messaging interface components. Then, they use these messages to store, further process or display different parameters such as the network topology information, network traffic status, the link quality indicator (LQI), received signal strength indicator (RSSI) etc. Moreover, they deduce inferences and accordingly draw charts from the management and the sensing data log files. These components can benefit from other external services to extend the visualization features of the tool such as requesting WEB-MAP services to display current geographical maps of the deployment terrains. Figure 7.2 shows the UML diagram of the visualization components. “ToolFrame” is the main container which holds all other visualization components. “DisplayPanel” holds all nodes and tracks their movements in the panel. “NodePainter” is the actual holder and the visualizer of the nodes information on the “DisplayPanel”. “Tabbedplugin” is the container where plug-ins are drawn.



Figure 7.2 UML diagram of the visualization classes inside the management tool

- **Events components**

There are many sources of events in the tool. A java marker interface is used to specify that all events derived from this marker interface are of the same type. In this case, all events in the tool are of type “ITEvents”. To specify the events more, we have added two additional abstract interfaces. The first class is “InEvent” which is base class of all internal events generated from the tool. As an example, the events that are generated by clicking on a node in the nodes panel “NodeClickedEvent”. The other class is “ExEvent” which is the base class of all external events coming from the nodes. An example of this class is the event that is generated when a new node is found in the topology “ENode”. Some of other events are also:

- receiving a sensing data packet “SensorValueEvent”
- receiving a management data packet from the network stacks “NetworkEvent”
- receiving a management data packet from the CPU scheduler “SchedValueEvent”
- Clicking on a node “NodeClickedEvent”

The Figure 7.3 shows an UML class diagram of the events which are signaled inside the tool.

- **Nodes data components**

These components are responsible for storing, displaying and processing the nodes information and their packets. The information of the nodes is of two types that are the sensing data generated by the sensors on the nodes and the management data. The framework provides the option that the sensed data and the management data can be stored either in user-specified or temporal XML files.

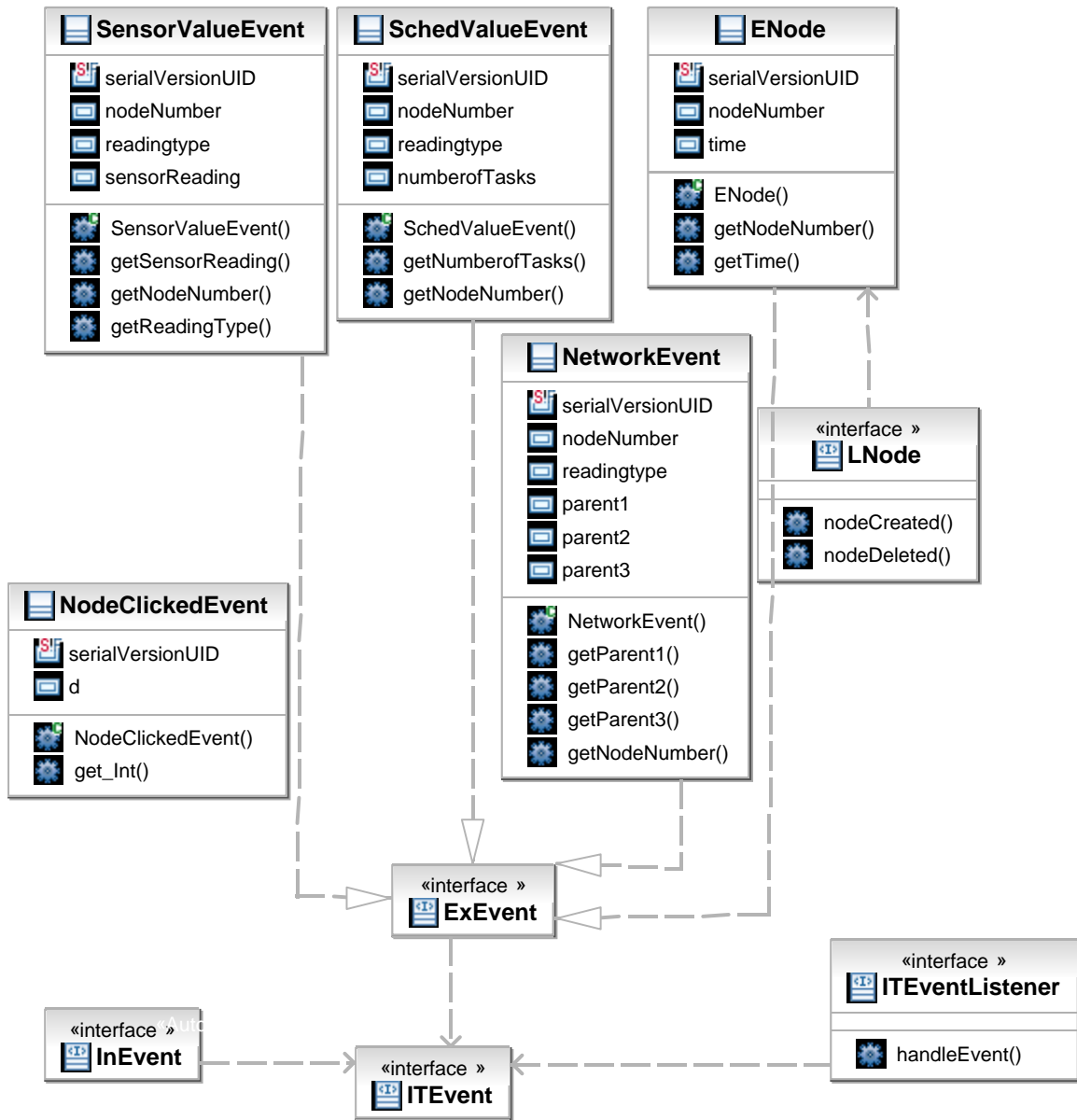


Figure 7.3 UML diagram of events classes inside the tool

7.2. *Plug-ins Implementation*

Figure 7.4 shows a UML class diagram of the plug-ins classes which are provided in the current implementation of the tool.

7.2.1. Plug-in Interface

The way to add additional plug-ins is the following. Firstly, the user has to extend the interface “Plugin” and implement its abstract functionality. This interface contains the following methods:

```
public abstract void set_Plugins(){  
    }  
public abstract void unset_Plugins() {  
    }  
public abstract void handle_Event(ITEvent event) {  
    }  
public void draw(Graphics2D graphics) {  
    }  
public void drawtotabs(Graphics2D graphics) {  
    }
```

The first method is internally called by the tool to initialize the plug-in, while the second method is called by the tool for de-initializing it.

The third function is periodically called by the tool to signal all plug-ins with all possible events. The argument of this function is an instance of a marker interface “ITEvent”, which represents all potential events in the tool. As described before, these received events are signaled either internally from the tool on the PC or externally from the nodes.

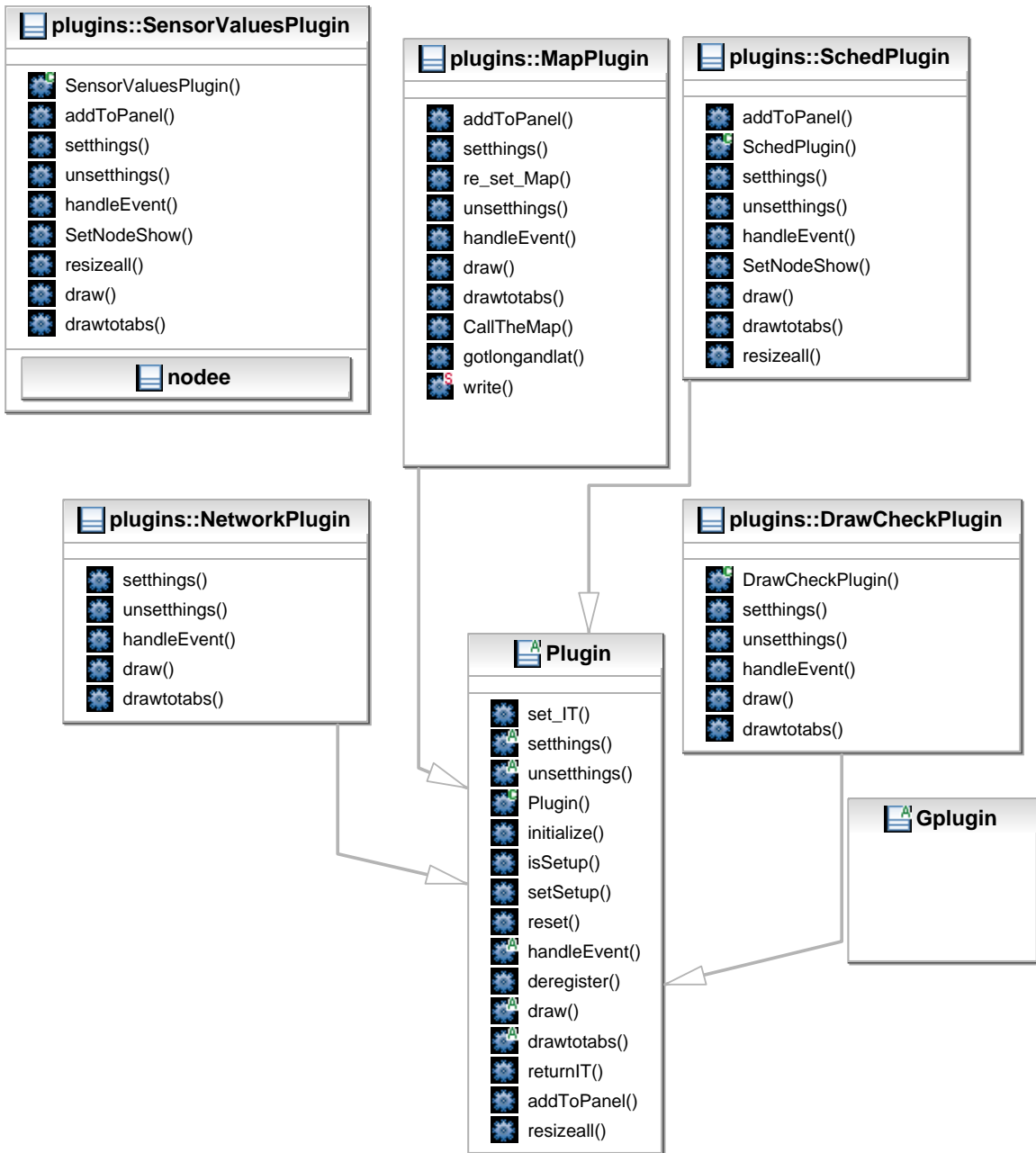


Figure 7.4 UML diagram of the plug-ins

The tool has two main panels for drawing. The first is for the nodes' deployment area that includes the nodes figures, the geographical map and information on the nodes' parameters such as the number of received management packets. The other panel is for the plug-ins. On this panel, the user can display all additional information of interest that are related to the plug-ins. The designer of a plug-in receives two "java.awt.Graphics" class instances for drawing on both the tool and the plug-ins panels. These are the arguments in the following two functions:

```
public void draw(Graphics2D graphics) {  
    }  
public void drawtotabs(Graphics2D graphics) {  
    }
```

These functions are internally invoked by the tool. By implementing these functions, the users' actual drawing is accomplished.

7.3. Main Classes of the Nodes-Side Components

The node-side components consist of the management entities which carry out the actual management on the nodes as well as the communication with the PC side components. These components can be seen as the management agents in the traditional network management such as in Simple Network Management Protocol (SNMP) [62]. The management entities specifications vary between the sensor nodes types depending on the applied hardware components and the application nature. The number of these management entities depends on the number of the modules which the user intends to manage or control. In our designed tool, the management entities are designed as modular “nesC” libraries. These libraries can be linked to the original program which is TinyOS operating system in our case. The implemented management entities (agents), which we have already designed, are following.

- **MAC management agent**

This agent is used to provide management information about the MAC layer. It is implemented in nesC and added as a library for TinyOS operating system. It is statically compiled and then linked before uploading the final binary file into the sensor node. In order to include this agent, the user only has to add this following line of code in the main application configuration file:

```
Main.StdControl -> MacC;
```

MacC is a configuration module, which is the top level component of this management agent. This module configures all the other libraries included by the application. During the deployment time, the user can remotely enable or disable the MAC agent via the tool on the PC side. For now, this agent collects the LQI and RSSI on each node and then sends them back to the tool on the PC, where the information is displayed on the screen and stored in XML files. Received Signal Strength Indicator (RSSI) and Link Quality Indicator (LQI) are significant parameters for several analyses such as the routing and positioning schemes. LQI measures the quality of the received signal, while RSSI indicates how strong the signal is.

There are two message types associated with this agent: control management message and data management message.

Figure 7.5 shows the structure of the control message (`BcastMsg`) and the data message (`MacMsg`). The control message is received by the agent on the node to enable or disable sending the data messages. This depends on the values of the (start) or (stop) fields. The field (seqnu) is used to mark the sequence of the messages for preventing loops and avoiding messages duplication. The message structure of the response messages (`MacMsg`) sent back from the agent contains some fields as shown in the figure. As mentioned in chapter two, nesC language uses the parameterized interface technique which is similar to the ports in TCP/IP network model. The parameterized interface with the number (151) is assigned for sending and receiving the control messages. Via this interface, the MAC management agent on the nodes is signaled to start or stop on sending data on the MAC. The interface numbered with (150) is used here to send the MAC data, which contains the RSSI and LQI fields in this case.

```
//The MAC messages parameters which are sent back from nodes to the base station.
typedef struct MacMsg {
    // The node for which we are measuring the RSSI and LQI
    uint16_t source; // Source of the message
    uint16_t org; // Originator of the message
    uint8_t RSSI; // RSSI value
    uint8_t LQI; // LQI value
    uint32_t seqnu; // sequence number
} MacMsg;
//The flooding messages which are sent for signaling nodes to start or stop sending data
about the MAC
typedef struct BcastMsg {
    uint8_t start; // Command Start
    uint8_t stop; // Command Stop
    uint16_t seqno;
} BcastMsg;
```

Figure 7.5 Messages structure of MAC control and data messages

From the memory perspective, we have measured the size in bytes of one of the management entities which is used to manage the MAC layer. The memory size is 802 bytes in the RAM and 1038 bytes in ROM.

Without MAC management layer, the required memory size in a node is:

36024 bytes in ROM - 7659 bytes in RAM.

With MAC management layer, the required memory size in a node is:

37062 bytes in ROM - 8461 bytes in RAM.

- **Network management agent**

This agent provides management information about the network. To use this agent, the user has only to add the following code line in the main configuration module of the application.

```
Main.StdControl -> NetworkC;
```

```
// The data management message of the network, which is sent back from nodes to the base station
// via the port "162"
typedef struct NTMsg {
//     uint16_t source; // The node for which we are measuring the rssi and lqi
    uint16_t org;
    uint16_t nb1;
    uint16_t nb2;
    uint16_t nb3;
    uint32_t seqno;
} NTMsg;
// The flood message, which is sent to signal nodes to start or stop sending management
// information about the Network via port "163".
typedef struct BNTMsg {
    uint8_t start; // Command Start
    uint8_t stop; // Command Stop
    uint16_t seqno;
} BNTMsg;
```

Figure 7.6 Network management agent messages structure

The user disables or re-enables this agent using the tool on the PC side. Figure 7.6 shows the structure of the control management message (`BNTMMsg`). This control message signals the agent to start or to stop sending data management messages. The data management message of the network (`NTMMsg`) contains the neighbors IDs in the routing table of the node. On the PC-side, this information of the data messages is stored in XML files. It is also used to display neighbors of the node and the routing tree on the tool display panel. The assigned parameterized interfaces are set as follows. The port (163) is set for exchanging the control management messages, while the port (162) is assigned for sending and receiving the data management messages from the nodes to the base node.

- **Scheduling management agent:**

This agent is used for monitoring the scheduler performance. The user has to add the following line of code in the main configuration file to include this agent:

```
Main.StdControl -> SchedC;
```

This agent keeps track of the number of the CPU-scheduling tasks which are left on the scheduler in each node. Then, it sends these values to the tool on the PC side, where they are displayed in graphs or charts forms on the screen and also restored in XML files. Figure 7.7 shows the structure of the control management messages (`BcastSchedMsg`) which are sent from the PC and received by the nodes to signal this agent. After signaling the agents, this agent starts issuing the data management messages (`SchedMsg`), whose message structure is also shown in the same figure. Parameterized interfaces are the following; port (170) is for the control management messages while (171) is for data management messages.

- **Application management agent**

It is used to control the application running on the nodes. To benefit from this agent, the user has to add the following line in the configuration component:

```

// The data management message of the CPU-scheduler, which is sent back from nodes to
the base station via the port "170"
typedef struct SchedMsg {
    uint16_t source; // Source
    uint16_t org; // Originator
    uint16_t nt; // this stands for the tasks number
    uint32_t seqnu; // Sequence numbers
} SchedMsg;
// The flood message, which is sent to signal nodes to start or stop sending management
information about the CPU-scheduler via port "171".
typedef struct BcastSchedMsg {
    uint8_t start; // Command Start
    uint8_t stop; // Command Stop
    uint16_t seqnu;
} BcastSchedMsg;

```

Figure 7.7 Scheduling agent messages structure

Main.StdControl -> FloodStartStopC;

From the tool on the PC side, this agent allows the user to on-line enable or disable the functioning of the nodes. This can be done by flooding the network with the control messages (BcastStartStop) that have the structure shown in Figure 7.8. This message contains three fields as shown in the figure.

```

// The flood message, which is sent to signal application on the nodes to start or stop
sending the sensing information about the via port "191".
typedef struct BcastStartStop {
    uint8_t start; // Command Start
    uint8_t stop; // Command Stop
    uint16_t seqno;
} BcastStartStop;

```

Figure 7.8 Messages structure of the management agent for stopping and functioning the nodes

Chapter 8

Evaluation of the Management Framework

In this chapter, the use of the features supported by the tool is elaborated. A real life demonstration of the tool is described. Then, the evaluation of these features is provided. This chapter is organized as follows. At first, the specification of the wireless sensor network, which we have used in the real life scenario, is described. Afterwards, the following issues are discussed:

- The usage of the tool to get and save the sensor samples
- CPU scheduling management using the tool
- Benefits of the geographical map services provided by the tool
- Network and node management
- Use of the tool services as a WEB application

8.1. Experiment Setup

A network of wireless sensor nodes that consists of ten nodes is set up. These nodes are called Tmote Sky from Sentilla Corporation (earlier moteIV [13]). The key features of each node are the following:

- MSP430 microcontroller
- Integrated Digital to Analog Converters (DAC), Digital to Analog Converters (DAC), and Direct Memory Access (DMA) controller.
- Programming via USB
- Support for TinyOS
- Compliant with IEEE 802.15.4 standard with 250kbps for the data and 2.4GHz for the carrier

- 16 pins as expansion support for external extensions
- Onboard integrated antenna
- Integrated internal and external temperature sensors, humidity sensor, solar radiation sensor, and photo synthetically sensor.

In the scenario, the nodes are deployed on the same floor in our institute building. Immediately after the deployment, these nodes initiate a collection tree in which each node periodically collects the samples and forwards them to the root node. According to our settings, every node sends every four seconds a sample taken from every sensor that exists on the board of the node. These readings will then be forwarded via other nodes or directly to the base station depending on the topology map. Finally, all samples reach the base station which is responsible for collecting all the sensed data. The base station, in turn, forwards this data to a computer or to a PDA. The other role of the base station is that it receives the command packets from the computer and then it disseminates them to the sensor nodes via a dissemination protocol. The computing device (personal computer or PDA) represents the only access interface to the wireless sensor network through which the user can manage and control the nodes in the network.

8.1.1. Samples Aggregation

For the sensor network described in the above subsection, the tool displays the nodes existing in the topology. Then, the user can choose any node in the topology to visualize its samples.

The tool identifies the existence of a node through its ID sent with the sample messages. It can also identify which of the sensing parameters is sent through a type identifier. The following identifiers are assigned with each of the sensed parameters: (humidity = 1, external temperature = 2, internal temperature = 3, illumination = 4, voltage = 5).

After the tool receives the packets coming from the deployed nodes, the tool processes the received raw data to find the actual sensing values. The user optionally saves the sensing data coming from the nodes to either temporal or to specific data files. The format of the XML file of the sensing data is shown in Figure 8.1. The elements in this XML file are the nodes associated with their attributes. The attributes are the timestamp

at which the tool receives the samples, and the sequence of every sample. The sub-elements of each of the nodes contain the different sample types and their values.

The tool monitors health of the deployed nodes by keeping track on the rates at which each node is working. This would provide a feedback whether the network is running at a certain quality of service (QoS). However, the user has to initially specify the sampling rate of the nodes.

Moreover, to generate the graphs representing the sensed data by parsing the XML files, the user has only to click on the node in the tool panel to generate its sensing graphs. Then, the user chooses the sensed parameters by clicking on the button associated with it. By doing this, the samples saved in the XML file are parsed and the data associated with this chosen node, with respect to the timestamp, will be displayed on the screen.

```
<?xml version="1.0" ?>
- <SensorReading>
- <Node ID="1" Time="0" index="1">
  <Volt>0.96569824</Volt>
</Node>
- <Node ID="0" Time="0" index="1">
  <InternalTemp>34.40966</InternalTemp>
</Node>
- <Node ID="5" Time="2" index="2">
  <Volt>0.8983154</Volt>
</Node>
- <Node ID="1" Time="0" index="1">
  <Humidity>15.382548</Humidity>
</Node>
- <Node ID="4" Time="0" index="1">
  <Illumination>439.45312</Illumination>
</Node>
- <Node ID="0" Time="4" index="1">
  <Volt>1.0704346</Volt>
</Node>
- <Node ID="5" Time="3" index="2">
  <Humidity>13.97511</Humidity>
</Node>
```

Figure 8.1 XML file of the stored sensing parameters

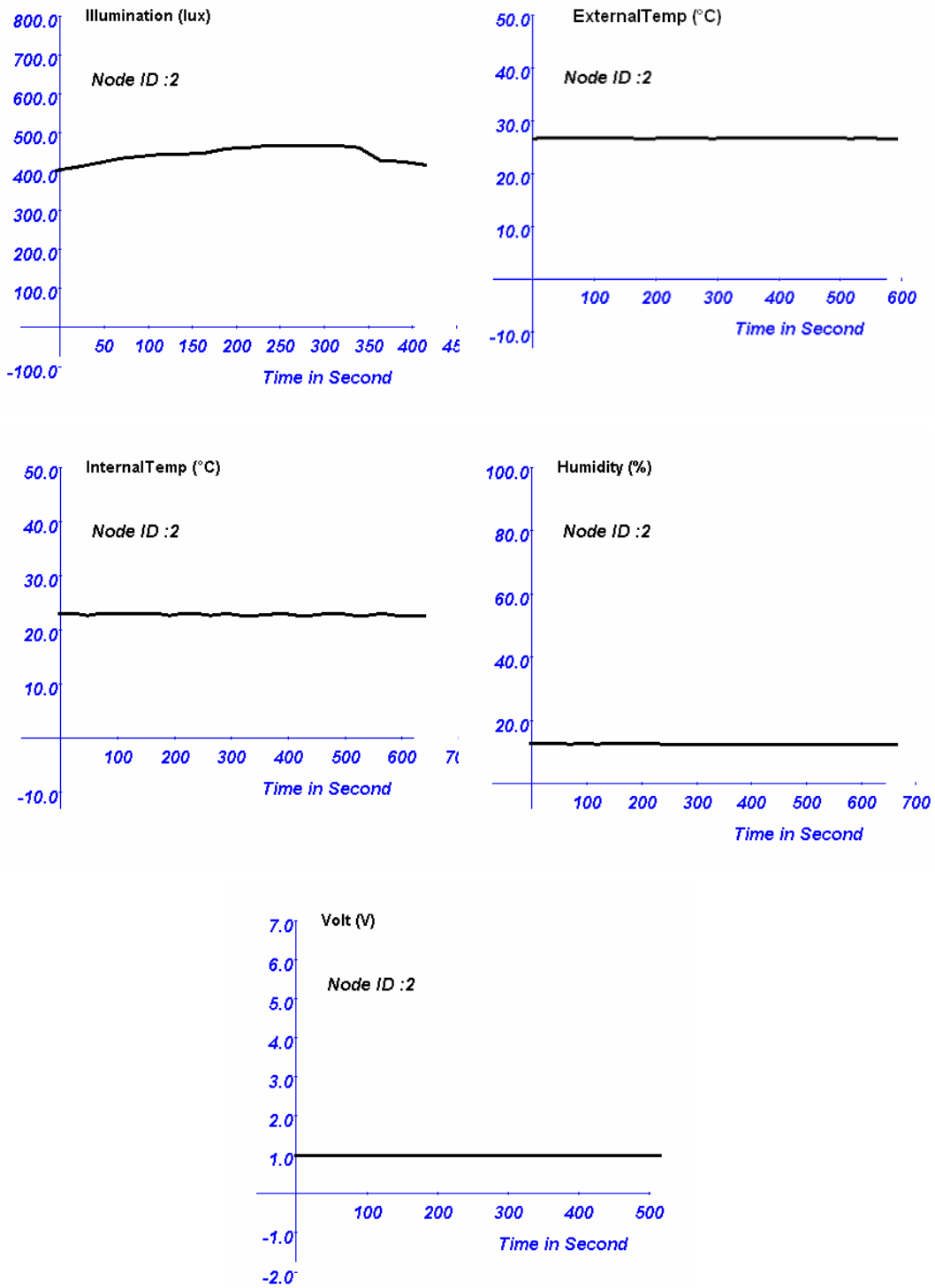
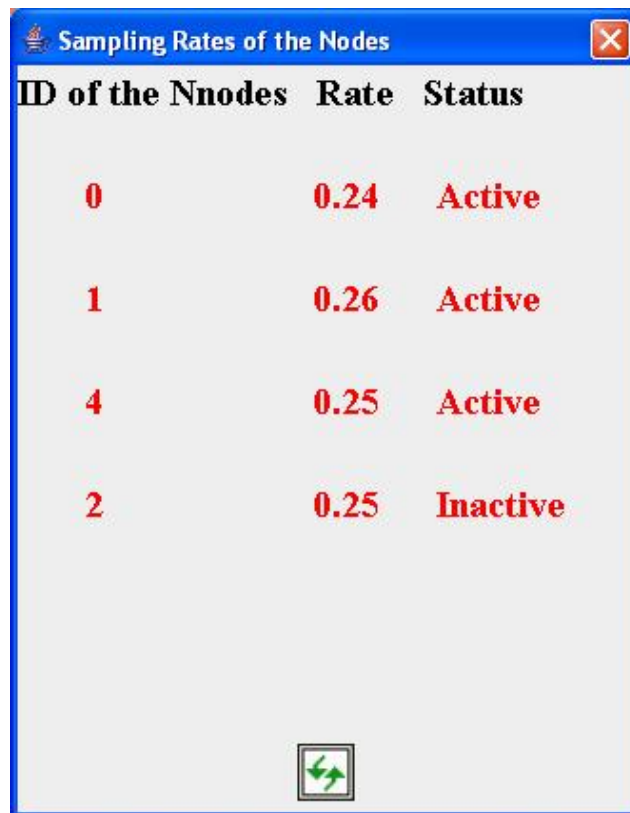


Figure 8.2 charts of samples of a sensor node generated by the tool

In Figure 8.2, few generated graphs for the following sensed parameters: illumination, external and internal temperature, humidity and volt. All these charts are automatically scaled to the screen size by the tool.

From the tool, we can remotely send the nodes into sleep mode and wake them up again. In this case, the nodes will stop on sending their sample data; however, they keep on running their management services in the background.

The framework provides sample rate tracking facility for observing the sampling rate values and the nodes status, whether each node is active or inactive. This facility initially requires two values to be set up by the user. The first one is the time during which the node is considered as inactive. If there are no messages received from a node during this reference time, then this node is set as an inactive node. The other reference is the number of packets at which the facility re-sets the averaging calculation of the received samples and re-starts averaging the sampling rate value. This is important to have always recent accurate values. A snapshot of this facility GUI is shown in Figure 8.3. The nodes IDs are listed first. Then, the sampling rate and the nodes status are shown for each node.



ID of the Nnodes	Rate	Status
0	0.24	Active
1	0.26	Active
4	0.25	Active
2	0.25	Inactive

Figure 8.3 Sampling rate dialog box

8.1.2. CPU Scheduling Management and Performance Analyses Using the Management Tool

As mentioned in the second chapter, the communication and the computation are the two main consumers of energy in WSNs. The energy consumption by the computation is mainly correlated with the CPU utilization. Evaluation of the CPU utilization in WSNs is less investigated by the researchers. Some of the works in this regard is what we have accomplished in [1], [2], [8].

The CPU scheduling performance analyzer is designed as an additional plug-in. Data regarding the CPU scheduler performance is collected from each node. Then, it is directly displayed as graphs. Additionally, this data is saved in XML format, shown in Figure 8.4, so that it can be used by the tool and by other XML-based applications.

```

<?xml version="1.0" ?>
- <SensorReading>
- <Node ID="1" Time="0" index="1">
  <Scheduling>173.0</Scheduling>
</Node>
- <Node ID="0" Time="0" index="1">
  <Scheduling>470.0</Scheduling>
</Node>
- <Node ID="1" Time="7" index="2">
  <Scheduling>167.0</Scheduling>
</Node>
- <Node ID="0" Time="7" index="2">
  <Scheduling>440.0</Scheduling>
</Node>
- <Node ID="1" Time="15" index="3">
  <Scheduling>164.0</Scheduling>
</Node>
- <Node ID="0" Time="15" index="3">
  <Scheduling>470.0</Scheduling>
</Node>
- <Node ID="1" Time="23" index="4">
  <Scheduling>177.0</Scheduling>
</Node>

```

Figure 8.4 XML file of CPU utilization data stored by the tool

Several observations have been demonstrated by this Plug-in. It is figured out that the transition in the computational load during the nodes' deployment is an important indicator of the network behaviors. For demonstration, different performance analyses have been performed using the tool.

Figure 8.5 shows the wireless sensor network which we have deployed to use with our tool. This network has the nodes 2, 3, 4, 5, 10, and 0. All these nodes are only one-hop away from the base station 0 and, none of these nodes is sending its packets via any other node. In this experiment, the number of the active nodes is gradually decreased by two every time after every certain time. During that, the scheduler performance of the node 2 was monitored. The tool has generated the CPU-scheduler performance graph shown in Figure 8.6. In this figure, it can be observed that how the number of the neighboring nodes affects the CPU-load of the node 2. Initially, the load is 220 Tasks/Second. After disabling 3 and 4, it is 190 Tasks/Second. Finally, the load is 160 Tasks/Second after disabling the nodes 5 and 10. Such analysis of the CPU-scheduling load provides a good measure of the node-positions in the topology and the network density at these positions. In other words, the congestion of the medium around a node can be indicated by the CPU load and we can measure it exactly by tools such as our designed tool.

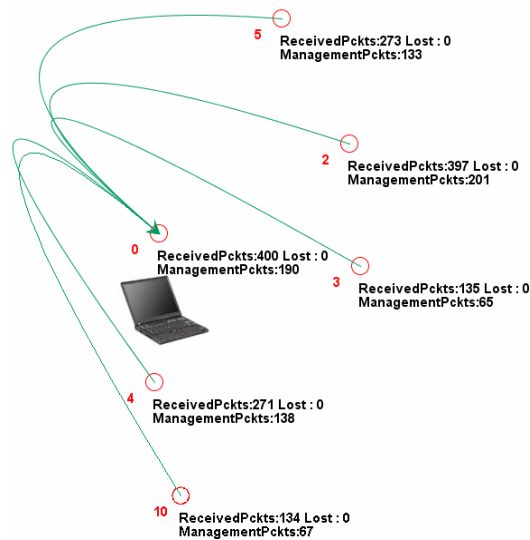


Figure 8.5 Topology of the network for demonstrating the CPU-scheduler management feature

The second experiment is the measurement of the CPU utilization. In this scenario, it is measured how much the CPU load decreases in both active (during messages transmission) and idle mode. In Figure 8.7-A, it is observed that the number of tasks is around 190 if the node is sending packets while it is 150 if the node is in idle mode. The experiment would also provide a measure of how much would be the energy consumption in both cases. During idle mode, the number of the tasks, which is 150 here,

represents the load caused by the background activities running on the node such as managing and maintaining the routes in the topology of the network.

The third experiment we have performed with the tool is measuring the effect of decreasing the number of sub nodes of a particular node. A significant decrease of the CPU utilization is observed as shown in the Figure 8.7-B. This figure shows the CPU utilization for a node that has four other additional sub-nodes. In this case the load is around 600 tasks. After disabling all of the sub-nodes, it is observed that the load drops to the value of 270 Tasks/Second. This indicates how much the load decreases while decreasing the number of the sub-nodes.

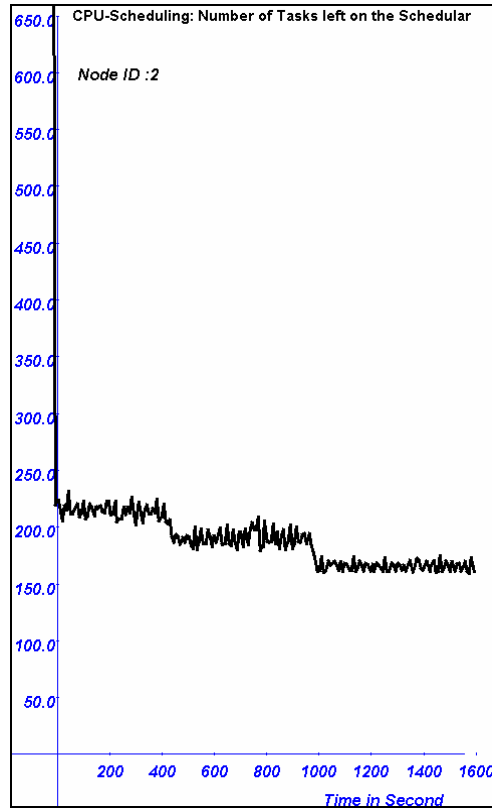


Figure 8.6 CPU-Scheduler performance while reducing the number of the neighboring nodes

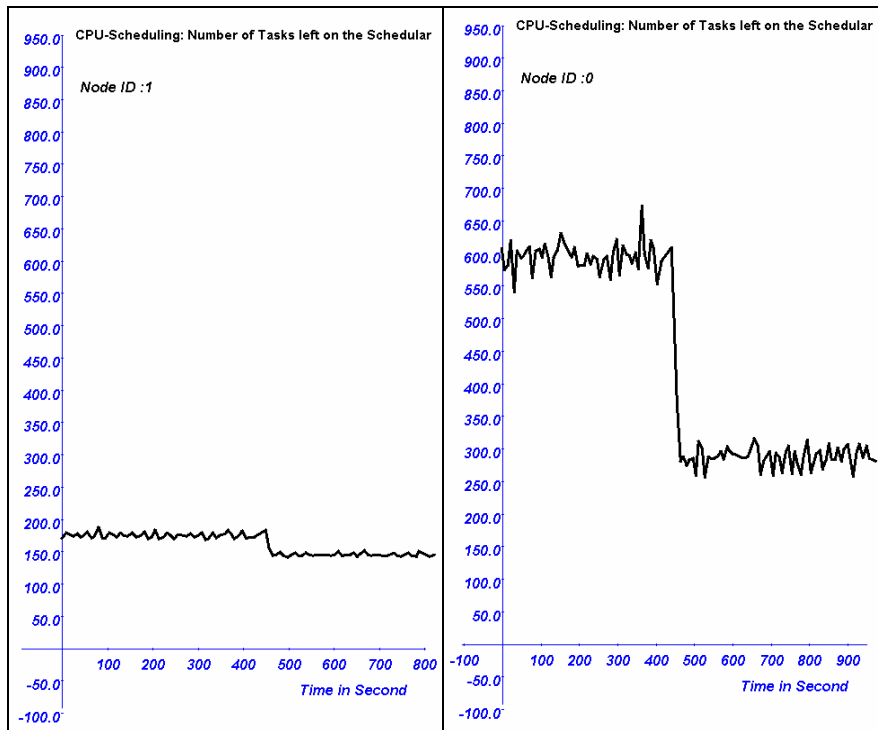


Figure 8.7 CPU-scheduler performance in two cases:
A-Left. Performance while functioning and while the node is disabled
B-Right. Disabling sub-nodes

8.1.3. Map Services and Localization

Localizing or positioning the nodes geographically is required in many WSN applications such as geographical routing, geographical attribute-based naming, and navigation systems. Localizing the nodes can be either absolute or relative. This depends on the used localization schemes and on the applied localization technologies. As mentioned earlier, this plug-in fetches the maps of the deployment area according to the parameters specified by the user (place name, address, latitude, longitude or zip). Adjusting the map on the panel is enabled by the tool, as the tool can shift the map left, right, down and up. Zooming in and out is also supported by the tool. Figure 8.8 shows the control panel of this plug-in.



Figure 8.8 Snapshot of the Map plug-in

Unfortunately, the nodes in the development kit that we have used with the management tool are not equipped with any kind of localization technologies such as GPS receivers. Therefore, the demonstration will only be limited to displaying a place of interest where the nodes can be potentially deployed. As an example, we assume a sensor network which runs a medical monitoring application at the University of Luebeck. The objective of this application is to monitor the vital signs of patients. To localize the patients carrying the sensor nodes in their actual positions on the map, our tool is capable of visualizing the deployment places. This is demonstrated as screen shot in Figure 8.9. In this figure, we can see how a map appears in the tool and how the nodes appear in their geographical positions on the map.

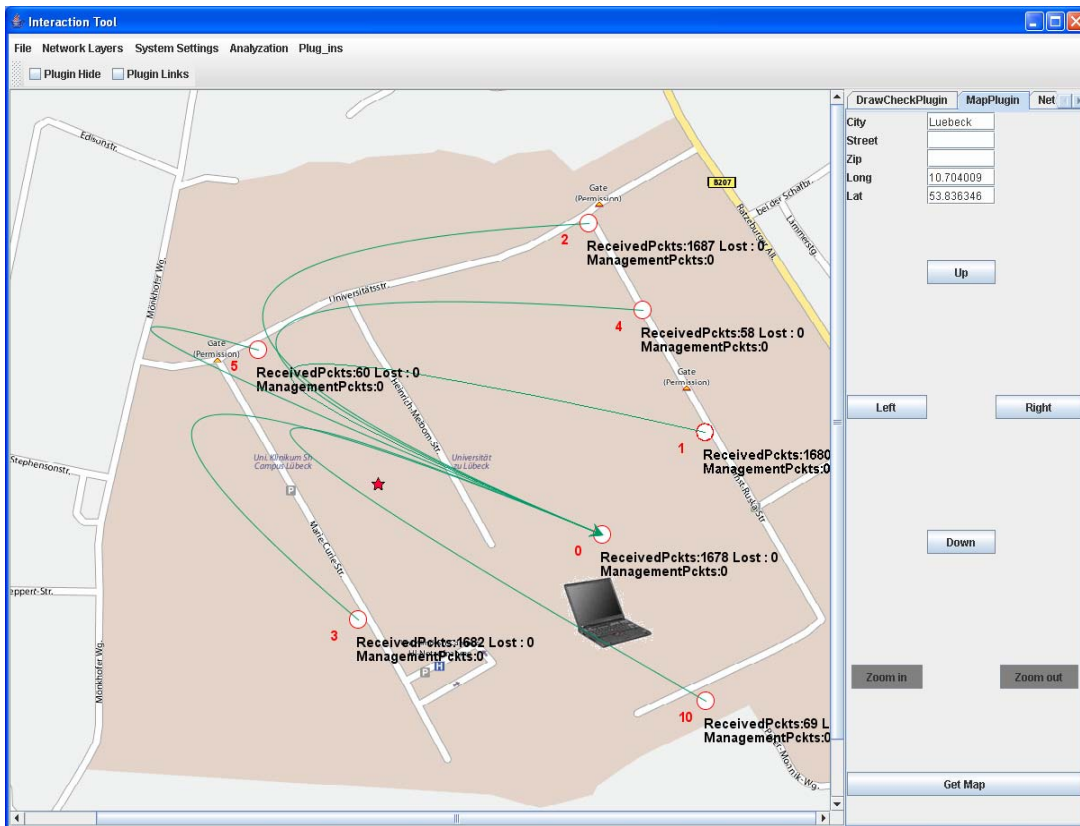


Figure 8.9, fetching the map of potential WSN deployment

8.1.4. Network Management

Inspecting the network parameters is required during the research on WSNs and during the application development. Our designed tool provides the possibility of exposing network parameters to the user of the WSN. In this subsection, it is described how the tool helps in monitoring the network activities.

Our tool is able to provide monitoring of the network activities at the multiple layers of the network stack. However, the current implementation supports the monitoring of significant parameters at the physical layer and the network layer.

To inspect the network layer, the user of the tool signals the nodes to start (or stop) sending information about their routing table contents. In response, the nodes start (or stop) transmitting their routing data to the base node. For demonstration, we have deployed a network that consists of four nodes whose IDs are 0, 6, 8, and 10. Then, we have signaled them from the PC to send their network information. In response, every node sends its current routing table contents. For the nodes which are not direct neighbors of the base station, this information will be routed to the base station by intermediate nodes. Then it is displayed as a table by the “NetworkPlugin”, snapshot is shown in Figure 8.10. In the table, the left most column is the node ID, while the other columns are the IDs of neighbors of this node. The first row shows that the node with ID 10 has 0, 8, and 5 in its routing table. The second column represents the current parent. The third and the fourth columns represent the second and third best potential parents of the node respectively. The special case in this table is the node 0. Node 0 is the base station node. It does not maintain its routing table because the only possible parent of this node is the PC connected to it.

The information is saved in XML data files as the Figure 8.11 shows. The tool saves for every received packet, the id of originator node, the timestamp at which the packet was received and the index of this packet. These parameters are added as attributes of the node. The XML sub-child element of a node element contains one of its neighbors.

<i>ID of Node</i>	<i>First Parent</i>	<i>Second Parent</i>	<i>Third Parent</i>
0	Base Station		
6	0	10	8
8	0	10	6
10	0	6	8

Figure 8.10 the routing tables taken from a screen shot from the NetworkPlugin

```

<?xml version="1.0" ?>
<SensorReading>
- <Node ID="6" Time="0" index="1">
  <Potential_Subnode>0.0</Potential_Subnode>
</Node>
- <Node ID="6" Time="0" index="1">
  <Potential_Subnode>10.0</Potential_Subnode>
</Node>
- <Node ID="6" Time="0" index="1">
  <Potential_Subnode>8.0</Potential_Subnode>
</Node>
- <Node ID="0" Time="1" index="1">
  <Potential_Subnode>126.0</Potential_Subnode>
</Node>
- <Node ID="0" Time="1" index="1">
  <Potential_Subnode>65535.0</Potential_Subnode>
</Node>
- <Node ID="0" Time="1" index="1">
  <Potential_Subnode>65535.0</Potential_Subnode>
</Node>
- <Node ID="8" Time="4" index="1">
  <Potential_Subnode>0.0</Potential_Subnode>
</Node>

```

Figure 8.11 An XML file shows the nodes and their parent nodes

In the current implementation, the tool fetches two important parameters from the physical layer. These parameters are RSSI and LQI. It displays these two parameters in the nodes panel near every node. As shown in Figure 8.12, the signal icon on the upper left corner of every node represents the strength of the RSSI, while the signal icon on the upper right corner of the node shows the LQI.

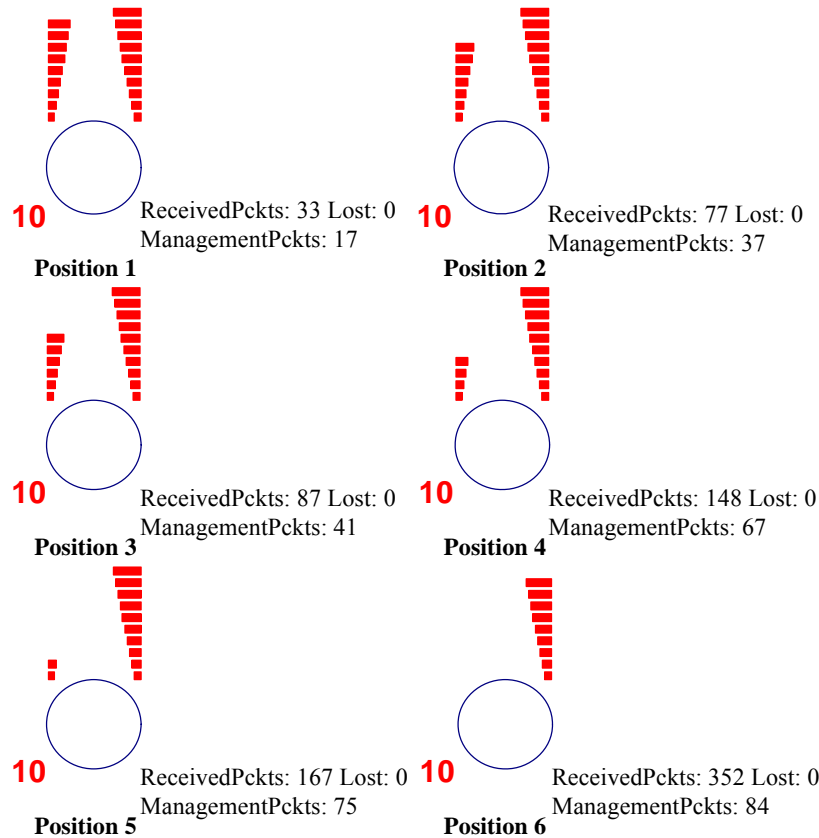


Figure 8.12 Snapshot of RSSI and LQI of a node while increasing the distance to the base station

Snapshots of a sensor node at six different positions have been taken while gradually increasing the distance between the node and the base station. Figure 8.12 shows how the signal strength degrades and becomes weaker when the distance of the node from the base station is being increased. In this figure, the position one is the closest and position six is the farthest from the base station.

8.2. WEB-Based WSN Viewer Application

The services which are provided by the designed tool can be used in the WWW (World Wide Web) applications. As the tool stores and deals with the data of the WSN in XML format, many XML-based web applications can exploit and manipulate this data. For example, by using web scripting languages such as PHP and JavaScript, this data can be provided as web services to the other users on different remote computers.

To demonstrate this feature, we have hosted the tool on a WEB server based on Apache HTTP Server 2.2.3. On this WEB server, we have implemented Web-based WSN viewer application based on PHP and JavaScript scripts. This application can dynamically and at the runtime parse the XML files which are generated by the tool and they also plot graphs that represent the information of the deployed sensor network. The plotted graphs expose the following information: the ID of each node existing in the topology and all different sensed parameters such as temperature, humidity, illumination and the voltage. They also expose all management information such as information about the network and the scheduler.

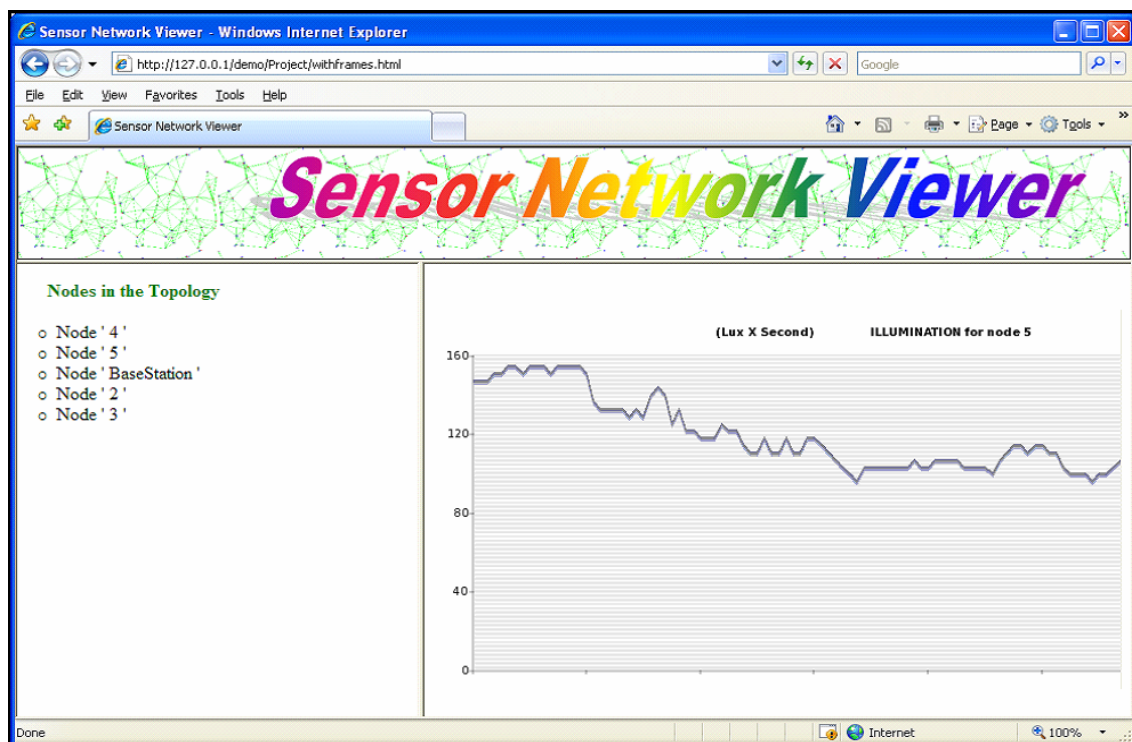


Figure 8.13 Sensor network viewer

On the browser side, the services provided to the user appear in an interactive graphical interface. This interface is the “Sensor Network Viewer” shown in Figure 8.14. The viewer has two main frames. In the left frame, shown in more detail in Figure 8.14, all nodes in the topology are automatically discovered and then displayed as a folded tree. By clicking on any nodes name in this frame, two main sub menus will be displayed. These are “Sample Readings” and “Management Readings”. By choosing the “Sample Readings”, different sensed parameters of the chosen node are expanded out as buttons such as temperature, illumination .etc. By choosing “Management Readings”, a menu contains the management parameters such as CPU utilization appears. If the user clicks on a button, the PHP scripts parse the XML files and accordingly generate a graph of that parameter for the chosen button. For example, by clicking on Nodes ‘5’ and then clicking the button illumination, the figure shown in Figure 8.15 is generated.

Nodes in the Topology

- Node ' 4 '
- Node ' 5 '
 - Sample Readings
 - Humidity in 5
 - External_Temp in 5
 - Volt in 5
 - Internal_Temp in 5
 - Illumination in 5
 - Management Readings
- Node ' BaseStation '
- Node ' 2 '
- Node ' 3 '

Figure 8.14 **Sensor network viewer left frame**

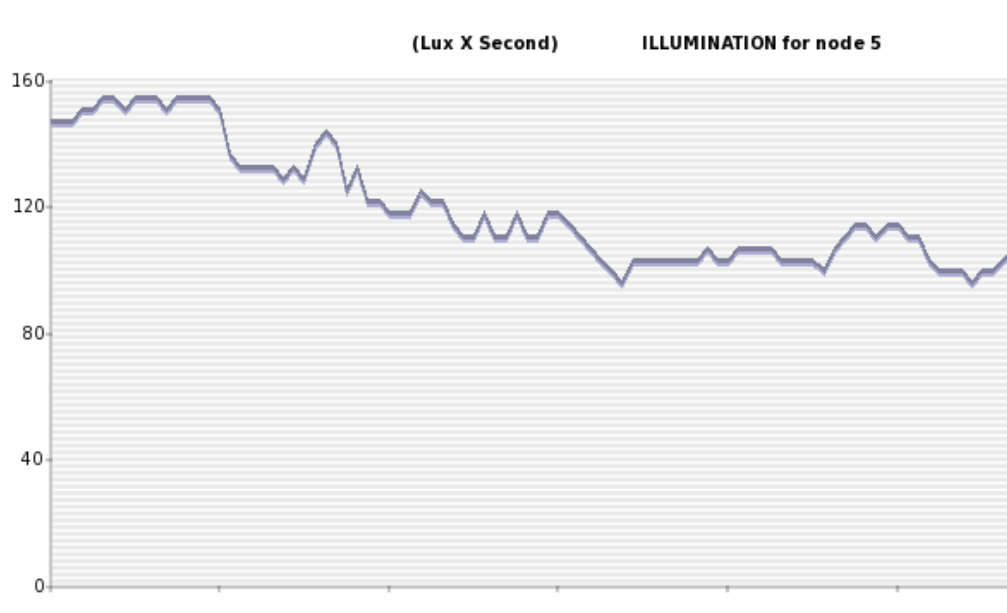


Figure 8.15 Chart produced by the sensor network viewer

Chapter 9

Conclusion

One of the main factors in designing applications or conducting research in wireless sensor networks is to have an efficient and integrated management environment. Furthermore, Quality of Service (QoS) can not be guaranteed if the applications providing the services are not manageable or are not interactive. In this dissertation, we presented a tool that comprises a management framework. It provides interaction paradigms for managing and monitoring the nodes in wireless sensor networks. To test the viability of the tool and its management models, we have designed it and validated it via experiments in real life scenarios. In the conclusion, a summary and the key contributions of this dissertation and, future work are mentioned.

9.1. A New CPU Scheduling Scheme

The initial work described in this dissertation is a new CPU scheduling scheme. This scheme enhances the performance in terms of fairness of the CPU-scheduler in a sensor node. Analytical proofs are deduced using our own specially designed simulator.

9.2. New Management Framework

A centralized management framework is proposed. This framework is for managing the nodes and interacting with them during the nodes' deployment. Furthermore, it provides an interface for the users to monitor the different parameters on the nodes. Moreover, it fetches the sensing information and it then displays it in auto-scaling graphs.

In the management of traditional networks, the management dimensions are: configuration, fault, performance, accounting and security. In our proposed management tool, we mainly targeted the configuration, performance and fault aspects.

9.3. Efficiency

The efficiency of the models supported by this tool to achieve the management is a key feature. Efficiency can be further enhanced when combining our management tool with other efficient communication schemes for WSNs such as data fusion, positioning algorithms and dissemination protocols.

9.4. Generality

In this relatively new area of research, having a generic architecture is still a difficult task due to the big diversity in wireless sensor network applications. In this dissertation, new suggestions for supporting the generality of WSNs are made. Some of the generality schemes are embedded into our design, which are the semantic methods, Message Interface generator (MIG) and Sensor net Protocol (SP). As another achievement in terms of generality, the tool deals with all information as XML data format, which is a step forward towards generalization, since many applications, especially WEB applications, deal with this format.

9.5. Architecture and Implementation

The architecture and the implementation of the management application were presented in this dissertation. Descriptions of both collaborative component groups of the tool are mentioned. These two groups of components reside on the node and on the computing machines.

9.6. Plug-ins

Plug-ins feature is another important feature supported by the tool. Using this feature, the tool can be extended by users according to their specific desires and their requirements.

As demonstration, we have presented the plug-ins which we have already designed.

These are the following:

- Sensing values Plug-in
- CPU scheduler Plug-in
- Network Layer Plug-in
- Map Plug-in

9.7. Real World Evaluation

In this dissertation, we have discussed the real life scenarios of the tool. These scenarios have been conducted to have evaluation of the tool during nodes' deployment. Different capabilities of this tool are shown such as:

- Displaying the sensing values of the multiple sensors
- Interacting with the CPU scheduler to fetch information about the CPU scheduling
- Using MAPs by the tool
- Interacting with nodes to monitor some parameters in the network stack
- Stopping or starting nodes remotely from the tool

As an extension of the evaluation, we have proved the generic nature by designing a WEB application. This WEB application is based on the XML data files generated by the tool at run time. Via this WEB application the tool's information is provided as HTML internet services.

9.8. Future Potential Work

Information provided by the tool can be considered as raw information. This raw information can further be processed and analyzed to get smart decisions for optimizing the WSNs performance. These smart decisions can potentially provide the best QoS solutions for WSNs. Our tool can be considered as a base of such solutions.

References

- [1] F. Tirkawi and S. Fischer, “Routing Sensitive Priority Scheduling”, in Proceedings of the ACM Workshop on Sensor and Actor Networks (SANET 2007).
- [2] F. Tirkawi and S. Fischer, “Operating System for Wireless Sensor Networks”, Master thesis at the university Lübeck, Institute for Telematics 2006.
- [3] J. A. Stankovic, Q. Cao, T. Doan, L. Fang, Z. He, R. Kiran, S. Lin, S. Son, R. Stoleru, A. Wood, “Wireless Sensor Networks for In-Home Healthcare: Potential and Challenges”, in Proceedings of the Workshop on High Confidence Medical Device Software and Systems (HCMDSS 2005).
- [4] F. Tirkawi and S. Fischer, “Remote Interaction Tool for Wireless Sensor Networks”, in Proceedings of the IEEE International Symposium on Wireless Pervasive Computing (ISWPC 2008).
- [5] F. Koushanfar, M. Potkonjak, and A. Sangiovanni–Vincentelli, “Fault-tolerance techniques for sensor networks”, in Proceedings of the. IEEE Sensors, June 2002, vol.2.
- [6] Z. Shelby, C. Pomalaza-Raez, and J. Haapola, “Energy optimization in multihop wireless embedded and sensor networks”, in Proceedings of the IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2004).
- [7] V. Subramonian, H. Huang, and S. Datar, “Priority Scheduling in TinyOS: A Case Study”, Washington University Technical Report WUCSE-2003-74.

- [8] F. Tirkawi and S. Fischer, “Adaptive Tasks Balancing in Wireless Sensor Networks”, in Proceedings of the International Conference on Information & Communication Technologies: from Theory to Applications (ICTTA’08).
- [9] Webpage of Crossbow Technology Inc.: [http:// www.xbow.com/](http://www.xbow.com/).
- [10] Webpage of TinyOS: <http://www.tinyos.net/> .
- [11] J. W. Hui and D. Culler, “The dynamic behavior of a data dissemination protocol for network programming at scale”, in Proceedings of the International Conference on Embedded Networked Sensor Systems (SenSys 2004).
- [12] Start Guide of Tmote nodes <http://www.moteiv.com/products/docs/tmote-sky-quickstart.pdf>.
- [13] Webpage of Moteiv Corporation: <http://www.sentilla.com/>.
- [14] Nucleus, G. Tolle, D. Culler, “Design of an Application-Cooperative Management System for Wireless Sensor Networks”. Proceedings of the European Workshop on Wireless Sensor Networks (EWSN 2005).
- [15] Webpage of <http://www.symbian.com/>.
- [16] Webpage of Scatterweb: http://www.inf.fu-berlin.de/inst/ag-tech/scatterweb_net/hardware/esb/index.html.
- [17] Webpage of PalOS: <http://sourceforge.net/projects/palos/>.
- [18] Webpage of Contiki operating system: <http://www.sics.se/contiki/> .
- [19] Webpage of Smart Dust: <http://www-bsac.eecs.berkeley.edu/archive/users/warneke-brett/SmartDust/> .

- [20] Webpage of Networked Embedded System Technology: <http://webs.cs.berkeley.edu/nest-index.html> .
- [21] Webpage of Center for Embedded System and Networking Sensing: <http://www.cens.ucla.edu/> .
- [22] Webpage of Berkeley WEBS: <http://webs.cs.berkeley.edu/> .
- [23] J. Hill, R. Szewczyk, A. Woo, S Hollar, D. Culler, and K. Pister, “System Architecture Directions for Network Sensors”, in Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS2000).
- [24] Webpage of Smart-Its project: <http://www.smart-its.org/> .
- [25] Webpage of sun Microsystems laboratories: <http://www.sunspotworld.com>.
- [26] Webpage of Institute for Telematics, University of Luebeck, Germany. www.itm.uni-luebeck.de .
- [27] Webpage of MANTIS operating system: <http://mantis.cs.colorado.edu/index.php/tiki-index.php> .
- [28] J. M. Rabaey, M. J. Ammer, J. L. Jr. da Silva, D. Patel, et al, “PicoRodio supports ad hoc ultra-low power wireless networking”, IEEE Computer Magazine, July 2000, vol.33, (no.7).
- [29] T. Eicken, D. Culler, S. C. Goldstein, and K. E. Schauer. “Active Messages: a mechanism for integrated communication and computation”, in Proceedings of the Annual International Symposium on Computer Architecture (ISCA 1992).

- [30] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, and I Stoica. “A unifying link abstraction for wireless sensor networks”, in Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys 2005).
- [31] A.Y.Wie, J.Heidemann, and D.Estrin, “Energy-Efficient MAC Protocol for Wireless Sensor Networks”, Proceedings of the IEEE Conference on Computer Communication and Networking (INFOCOM 2002).
- [32] Webpage of TinyOS: http://www.tinyos.net/presentations/Boot_Camp/.
- [33] G. Pankaj, Deadline scheduler for TinyOS. http://www.isi.edu/~weiye/teaching/cs558sm04/selected_projects.html.
- [34] Ph. Levis, and N. Lee, TOSSIM (A Simulator for TinyOS Networks) September 17, 2003.
- [35] A. Clemm, Network Management Fundamentals, Published by Cisco Press 2006.
- [36] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin. “Sympathy for the sensor network debugger”, in Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys 2005).
- [37] D. Malan, T. Fulford-Jones, M. Welsh, and S. Moulton “CodeBlue: An Ad Hoc Sensor Network Infrastructure for Emergency Medical Care”, in Proceedings of the International Workshop on Wearable and Implantable Body Sensor Networks, (BSN 2004).
- [38] L. B. Ruiz, J. M. S. Nogueira, and A. A. Loureiro. MANNA: “A management architecture for wireless sensor networks”, IEEE Communications Magazine, 41(2):116-125, February 2003.

- [39] M. A. M. Vieira et al., “Scheduling Nodes in Wireless Sensor Network: A Voronoi Approach”, in Proceedings of the Annual IEEE International Conference Local Computer Networks, (LCN 2003).
- [40] B. Deb, S. Bhatnagar, and B. Nath, “A topology discovery algorithm for sensor networks with applications to network management”, in Proceedings of the International Workshop on Wireless Communications and Networking (WCNC 2002).
- [41] L. B. Ruiz, T. R. M. Braga, F. A. Silva, H. P. Assunção, J. M. S. Nogueira, and A. A. F. Loureiro. “On the design of a self-managed wireless sensor network”, IEEE Communications Magazine. July 2005. vol. 43, pp. 95-102.
- [42] nesC Language Reference Manual.
- [43] L. Ruiz, I. Siqueira, L.B. Barbosa, H.C. Wong, J.M. Nogueira, A.A.F. Loureiro. “Fault Management in Event-Driven Wireless Sensor Networks”, in Proceedings of the International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM'04).
- [44] M. Ringwald, M. Cortesi, K. Römer, A. Vitaletti: “Demo Abstract: Passive Inspection of Deployed Sensor Networks with SNIF”, in Proceedings of the European Conference on Wireless Sensor Networks (EWSN 2007).
- [45] M. Ringwald, M. Yücel, K. Römer: “Demo Abstract: Interactive In-Field Inspection of WSNs”, in Proceedings of the European Conference on Wireless Sensor Networks (EWSN 2006).
- [46] C. Intanagonwiwat, R. Govindan, and D. Estrin. “Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks”, Proceedings of the International Conference on Mobile Computing and Networking (MobiCom 1999).

- [47] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu. “The broadcast storm problem in a mobile ad hoc network”, in Proceedings of the International Conference on Mobile Computing and Networking (MobiCom 1999).
- [48] T. Siok Kheng M. Alistair: “Adaptive Probabilistic Epidemic Protocol for Wireless Sensor Networks in an Urban Environment”, in Proceedings of International Conference on Computer Communications and Networks, (ICCCN 2007).
- [49] J. G. Jetcheva and D. B. Johnson. “Adaptive Demand-Driven Multicast Routing in Multi-Hop Wireless Ad Hoc Networks”, in Proceedings of Mobile Ad Hoc Networking and Computing (MobiHoc 2001).
- [50] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: “A Tiny Aggregation Service for Ad-Hoc Sensor Networks”, in Proceedings of the fifth OSDI, December 2002.
- [51] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. “Synopsis diffusion for robust aggregation in sensor networks”, in Proceedings of Embedded networked sensor systems (SenSys 2004).
- [52] R. Fonseca, O. Gnawali, K. Jamieson, S. Kim, P. Levis, and A. Woo. The Collection Tree Protocol (CTP) <http://www.tinyos.net/tinyos-2.x/doc/html/tep123.html>
- [53] Webpage of: <http://www.tinyos.net/tinyos-1.x/doc/nesc/mig.html>.
- [54] Webpage of www.iso.org, standards under development.
- [55] Webpage of www.ietf.org/, IPv6 over Low Power WPAN.
- [56] Webpage of www.hartcomm2.org

- [57] Webpage of www.zigbee.org

- [58] Webpage of www.wibree.com

- [59] S. Hadim, N. Mohamed, “Middleware: middleware challenges and approaches for wireless sensor networks” 2006 Published by the IEEE Computer Society Vol. 7, No. 3; March 2006.

- [60] SerialForwarder v 1.1 Bret Hull – October 10, 2001.

- [61] P. Levis, N. Patel, D. Culler, and S. Shenker, “Trickle: A self-regulating algorithm for code maintenance and propagation in wireless sensor networks”, in Proceedings of ACM Symposium on Network Systems Design and Implementation (NSDI 2004).

- [62] Webpage of CISCO www.cisco.com.

- [63] F. Benbadis, T. Friedman, M. Dias de Amorim, S. Fdida “GPS-Free-Free Positioning System for Wireless Sensor Networks”, in Proceedings of Wireless and Optical Communications Networks (WOCN 2005).

- [64] D. Pfisterer, and M. Lipphardt, C. Buschmann, H. Hellbrück, S. Fischer and J. H. Sauselin: “MarathonNet: Adding value to large scale sport events - A connectivity analysis”, in Proceedings of Integrated Internet Ad hoc and Sensor Networks (InterSense 2006)

- [65] R. Iyengar, B. Sikdar, “Scalable and distributed GPS free positioning for sensor networks”, website, Rensselaer Polytechnic Institute, Troy, NY, 2003.

- [66] W. L. Lee, A. Datta, and R. Cardell-Oliver “Network Management in Wireless Sensor Networks” Handbook of Mobile Ad Hoc and Pervasive Communications, American Scientific Publishers, 2006.

- [67] T. Gao et al. "Vital Signs Monitoring and Patient Tracking Over a Wireless Network", IEEE-EMBS 27th Annual International Conference of the Engineering in Medicine and Biology, Sept. 2005.
- [68] P. Levis and D. Culler, "Mat'è: a tiny virtual machine for sensor networks", in Proceedings of Architectural Support for Programming Languages and Operating Systems (ASPLOS 2008).
- [69] S. R. Madden, M.J. Franklin and J.M. Hellerstein , "TinyDB: An Acquisitional Query Processing System for Sensor Networks", ACM Trans. Database Systems, vol. 30, 2005.
- [70] C. Srisathapornphat , C. Jaikaeo and C. Shen , "Sensor Information Networking Architecture", in Proceedings of International Workshop on Parallel Processing, (ICPPW 2000).
- [71] D C. Fok , G. Roman and C. Lu , "Mobile Agent Middleware for Sensor Networks: An Application Case Study", in Proceedings of Information Processing in Sensor Network (IPSN 05).
- [72] K. Marzullo, "Tolerating Failures of Continuous-Valued Sensors", ACM Trans. Computer Systems, Vol. 8, No. 4, Pages 284-304, November 1990.
- [73] El M. Ould-Ahmed-Vall, G F. Riley and B. S. Heck, "A Geometric-Based Approach to Fault-Tolerance in Distributed Detection Using Wireless Sensor Network", in Proceedings of Information Processing in Sensor Network (IPSN2006).
- [74] J. Polastre, J. Hill, and D. Culler, "Versatile Low Power Media Access for Wireless Sensor Networks", in Proceedings of ACM Conference on Embedded Networked Sensor Systems (SenSys 2004).

- [75] Webpage of Telecommunication Standardization Sector (ITU-T) <http://www.itu.int/ITU-T/index.html>.
- [76] Webpage of the Distributed System Group <http://www.vs.inf.ethz.ch>.
- [77] Webpage of <http://www.tyndall.ie> .
- [78] F. Zhao, J. Liu, J. J. Liu, L. Guibas and J. E. Reich, “Collaborative Signal and Information Processing: An Information Directed Approach”, in Proceedings of the IEEE, Vol. 91, No. 8, Pages 1199-1209, Aug. 2003.
- [79] Webpage of <http://www.ietf.org/rfc/rfc1663.txt>.
- [80] F. Tirkawi, S. Fischer, “Generality Challenges and Approaches in WSNs”, International Journal of Communications, Network and System Sciences, Vol. 4 February, 2009.

Appendix A

Taxonomy of Wireless Sensor Networks applications:

Following is a brief description of the fields in which WSNs are currently being used or can potentially be used in future.

Environmental applications:

WSNs can be established for monitoring the environmental conditions whose observations are of a significant importance. Examples of such applications are: habitat monitoring (region surveillance, flood detection, forest fire detection), toxic zone monitoring, comprehensive studies of species behaviors, chemical detection, crops and livestock living conditions exploration. Most of the time, such applications of WSNs are characterized as low rate sampling WSN.

Industrial applications:

There is wide range of industrial applications in which WSN technology is the best choice for providing smart and flexible control. Such applications are resource mining, automotive (vehicle, aircraft, airspace ...etc.), different industry sectors (textile, metal), smart building and home automation. Setting actuators in such WSNs would additionally open a new dimension by permitting effective interaction possibilities. Such WSNs are characterized as high rate sampling applications.

Medical applications:

WSNs can be used in emergency situations, disaster relief, disease tracking (global malaria conditions tracking), and individual patient tracking and monitoring. The later one has been extensively targeted by researchers in many projects such as Codeblue[37] and Advanced Health and Disaster Aid Network (AID-N)[66]. The objective of these projects is to provide a WSN platform that can provide remote monitoring of the vital signs of patients and elderly people at hospitals as well as at their homes. Potential parameters that can be monitored are heart rate, blood pressure, oxygen blood saturation.

WSNs can replace many medical appliances which are used for providing electrocardiogram, electromyogram and electroencephalograph analyses. For example, in the project Codeblue a special sensor node is designed to provide the electromyogram which can be used to measure the uncontrolled shaking of Parkinson disease. This sensor node can be easily carried by the patient and it can provide a controlled fine-tuned dosage to the patient [37]. These applications run at high rate sampling.

Military applications:

WSN technology is very attractive for military scenarios due to its ad hoc nature and less cost. Moreover, WSNs can run in extremely hostile conditions. WSN can play a big role in the Command, Control, Communications, Computing, Intelligence, Surveillance, Reconnaissance, and Targeting (C₄ISRT) military systems. There are many studies of WSN in this area such as intrusion detection and border protection, shooter localization in urban terrain, and radiation monitoring. These applications can be classified as low rate WSN applications.

Commercial application:

WSN can provide a big aid for warehouse items controlling. If a node is attached to every item, controlling the inventory would be very fast and very efficient. Furthermore, the user can easily find out the accurate number and the location of the items. Product quality can also be tracked by the WSN such as expiry date, environmental conditions around the items. WSN can also be used in logistic applications because they can provide intelligent tracking transportation of the goods. Commercial applications are low rate WSNs applications.

Appendix B

Design Details of the CPU Scheduler Simulator

In this appendix, a brief description of the design and the functionality of the “CPU Scheduler Simulator” are presented.

EJS (Easy Java Simulations) framework is used for system modeling and providing a flexible component-based GUI design. EJS is a general tool for creating discrete computer simulations. It provides high level abstractions for perfectly modeling the complicated physical incidents. Simulators created through EJS run as stand-alone applications and they are platform independent as well. Moreover, EJS provides drag and drop feature to add components, which makes simulator development efficient and flexible. One key advantage of EJS is that it separates the logic, the data and the GUI.

“CPU Scheduler Simulator” simulates the task queues in a sensor node. These queues are event queue, high priority queue and low priority queue. In the Figure 9.1, the left most chart is the event queue (EQueue), the middle is the high priority queue (HPQueue) and the right most figure is the low priority queue (LPQueue). Moreover, “CPU Scheduler Simulator” displays the current real time of the simulator. In addition, it has two sliders. The first slider is for setting the number of sub nodes which forward their packets via the simulated node. The highest value of this slider is 1000 which means the simulator can provide the simulation up to 1000 sub-nodes. Mainly, the packets of these sub-nodes are utilizing the event and the high priority queues. The second slider represents the temporal priority switching ratio between the low and high priority tasks. This slider sets the point of each duty cycle at which the priority is switched. The duty cycle here is set to 2000 milliseconds. It also displays the exact numeric values of the low priority and high priority task queues. This simulator enables stopping, restarting the simulation and stepping it.

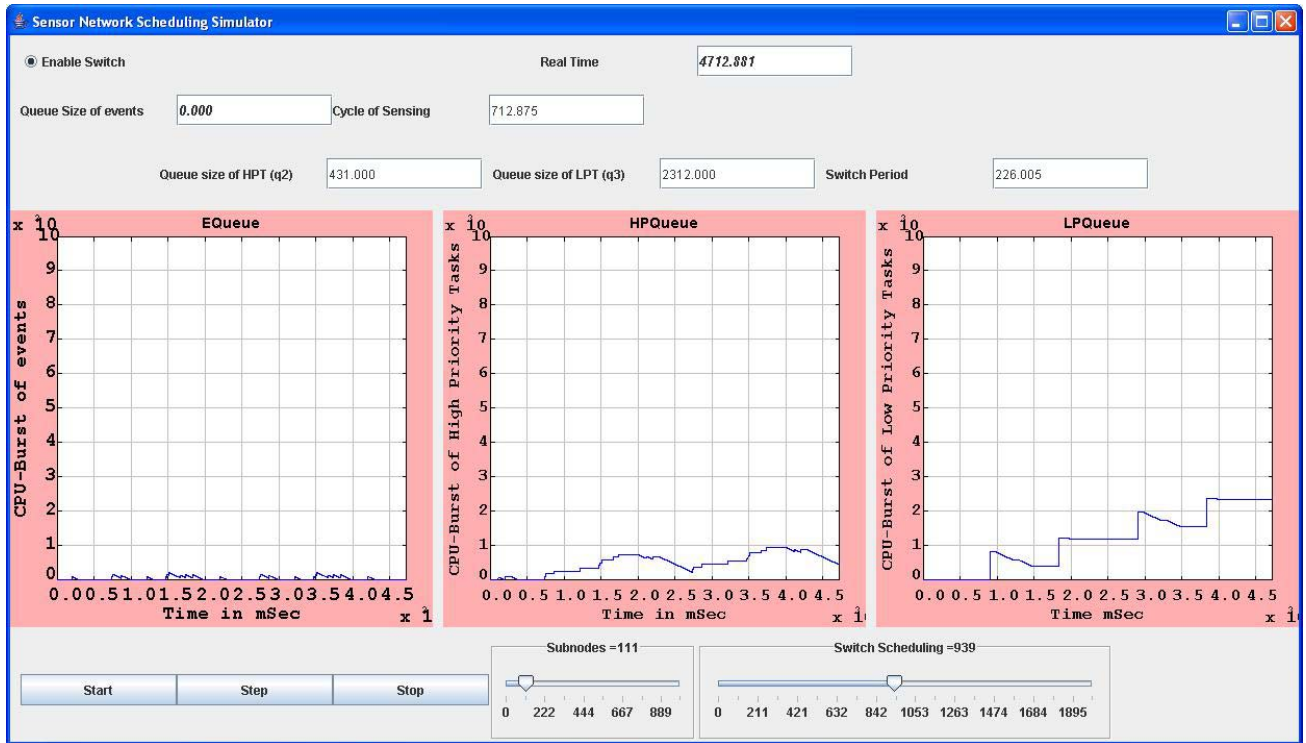


Figure 9.1 CPU Scheduler Simulator

In Table 9.1, all events, tasks coming from the sub-nodes, and the locally generated tasks are presented. These tasks are repeatedly triggered by the simulator at a normal random time. Then all these tasks will be accumulated on the corresponding queues. The source code of the function, which represents the execution flow of the queues in the simulator, is shown in Figure 9.2.

Events (q1)	Forwarding Tasks (q2)	Local Tasks (q3)
eventSend1	taskForward2	taskSense1
eventSense1	taskReceive2	taskProcess1
eventReceive1	taskRadio2	taskEncrypt1
eventTimer1	taskSend2	taskEncode1
eventRadio1		taskReceive1
eventForward2		taskSend1
eventReceive2		taskRadio1
eventSend2		
eventRadio2		

Table 9.1 Tasks and events modeled in the simulator


```
// One execution unit is reduced from the interrupt queue at each simulation step. It
// returns to check again if there are still interrupt tasks.
// q1 is the event queue which is served immediately in case it contains tasks.
if (q1 > 0) {
    q1 = q1 -1;
    return; }

// In case switching is not enabled: Execute tasks from the q2 which has the highest
// priority. After completion of q2, start executing q3 which has lesser priority.
if(!switch){
    if(q2>0 ){
        q2 = q2-1;
        return; }
    if(q3>0){
        q3 = q3-1;
        return; }
}

// In case switching is enabled. Execute tasks according to the enabled bq2 and bq3
// which depend on switching percentage of the duty cycle.
if(switch){
    if( ((bq2==true) || (q3==0)) && q2>0 ){
        q2 = q2 -1;
        return; }
    if( ( (bq3==true) || (q2 ==0 ) ) && q3>0 ){
        q3 = q3 -1;
        return; }
}
```

Figure 9.2 Execution flow of the queues.

VITA

February 23.1977 Born in Hama, Syria

2005 Thesis submitted for admitting to the PhD program at the University of Luebeck

1995 – 2000 Postgraduate Diploma in Electronics Engineering
Thesis (Solar Tracking System, Classification of modern automotive Electronics)

TECHNICAL SKILLS

Software and Tools Eclipse, NetBeans, MATLAB, Visio, Visual Studio, MS Office

Programming Java, C++, C, C#, nesC, OO Programming

Languages

Embedded Systems Wireless Applications, Keil Embedded, Development Tools, Microcontroller applications design, AVR Studio, MAP430 IAR Embedded Workbench

Databases MySql, MS Access

Operating Systems Windows 2000/NT/XP/Vista, UNIX

Web und server JSP, Servlet, DHTML, JavaScript, PHP, CSS, XML, Apache Technologies Server, Tomcat

Related Publication

- ❖ F. Tirkawi and S. Fischer, “Remote Interaction Tool for Wireless Sensor Networks”, in The 3rd IEEE International Symposium on Wireless Pervasive Computing 2008 (ISWPC'08), Santorini, Greece, 2008
- ❖ F. Tirkawi and S. Fischer, “Adaptive Tasks Balancing in Wireless Sensor Networks”, in The 3rd IEEE International Conference on Information & Communication Technologies: from Theory to Applications (ICTTA'08), Damascus, Syria, 2008
- ❖ F. Tirkawi and S. Fischer, “Routing Sensitive Priority Scheduling”, Poster in the first ACM workshop on sensor and actor networks, MobiCom, 2007, Montréal, Canada
- ❖ F. Tirkawi and S. Fischer, “Operating System for Wireless Sensor Networks”, Master thesis at the university Lübeck, Institute for Telematics 2005
- ❖ F. Tirkawi and S. Fischer, “Generality Challenges and Approaches in WSNs” International Journal of Communications, Network and System Sciences, Vol. 4 February, 2009.

