The ability of a Wireless Sensor Network (WSN) to monitor and capture real-time environmental data invites its use in several application domains ranging from the military to agriculture. Consequently, there is a lot of academic and industrial research happening in this field. In order to effectively support this extensive research, there is a need for WSN testbed that provides a platform for experimentation and subsequently supports the development of new WSN services.

The Web-based Wireless Sensor Network Testbed, Scibed, is an experimental testbed intended to provide an infrastructure for deploying and testing sensor network applications. The testbed will allow authorized users to reprogram a deployed network, read execution results, and query data from the database. Users will be able to access the testbed locally through the Web interface. The operations of the testbed are supported by a group of micaz sensor motes, which run on TinyOS operating system to communicate wirelessly.
# TABLE OF CONTENTS

Abstract ........................................................................................................ ii

Table of Contents ........................................................................................ iii

List of Figures .............................................................................................. vi

List of Tables .............................................................................................. viii

1. Introduction and Background ............................................................... 1
   1.1 Background ..................................................................................... 1
   1.2 Simulation ....................................................................................... 2
   1.3 Project Motivation .......................................................................... 3
   1.4 Related Work ................................................................................ 5
   1.5 Contributions of the Project ......................................................... 10

2. Wireless Sensor Network Testbed ......................................................... 11
   2.1 Project Scope ................................................................................ 11
   2.2 Project Objectives .......................................................................... 11
   2.3 Project Strategy ............................................................................ 12
   2.4 Project Stages and Outcomes ....................................................... 13
   2.5 Project Requirements ................................................................. 13
2.5.1 Functional Requirements........................................13
2.5.2 Non-functional Requirements.................................13

2.6 System Description.....................................................15
2.6.1 Users................................................................15
2.6.2 Web Interface.........................................................16
2.6.3 Database...............................................................16
2.6.4 Core System.........................................................16

3. System Design..........................................................17

3.1 Components of Proposed System.................................17
3.1.1 Hardware............................................................17
3.1.2 Operating System..................................................19
3.1.3 Sense: NesC Example Application...............................19
3.1.4 Deluge T2 – Data Dissemination Protocol.........................21
3.1.5 Serial Forwarder – Data Collection Component.................27
3.1.6 Software Framework...............................................29
3.1.7 GUI tools..........................................................31

3.2 Testbed Architecture................................................38
3.3 Discussion..............................................................39

4. Evaluation and Results................................................40

4.1 Evaluation............................................................40
4.2 Results...............................................................40
5. Future Work.................................................................42

6. Conclusion.................................................................43

Bibliography and References

APPENDIX A. PHP SCRIPTS AND PROGRAMS

APPENDIX B. SOFTWARE INSTALLATIONS

APPENDIX C. TESTBED SETUP INSTRUCTIONS
LIST OF FIGURES

Figure 2.1. Core System Components………………………………………………..16
Figure 3.1. Crossbow MICAz Mote……………………………………………………18
Figure 3.2. Crossbow MIB520 Interface Board…………………………………………18
Figure 3.3. Output of Compilation……………………………………………………..20
Figure 3.4. Output of Program Installation…………………………………………..21
Figure 3.5. Preparing the Base Station…………………………………………………..23
Figure 3.6. Preparing the Client Motes…………………………………………………24
Figure 3.7. Injecting Image File Into Base Station……………………………………25
Figure 3.8. Dissemination Command Sent to Network………………………………26
Figure 3.9. Installation of TestSerial Application……………………………………28
Figure 3.10. Displaying Packets Sent and Received ……………………………….28
Figure 3.11. Displaying Packets from Listen Tool…………………………………..29
Figure 3.12. Testbed Modules…………………………………………………………30
Figure 3.13. Index Page………………………………………………………………..32
Figure 3.14. Setup Page…………………………………………………………………33
Figure 3.15. Request Account Page……………………………………………………33
Figure 3.16. Login Page…………………………………………………………………34
Figure 3.17. Schedule Job Page………………………………………………………..34
Figure 3.18. Manage Jobs Page…………………………………………………………35
Figure 3.19. Contact Page………………………………………………………………35
Figure 3.20. Create Users Page………………………………………………………….36
LIST OF TABLES

Table 1.1. Comparison of Testbeds.................................................................9
Table 2.1. Project Stages and Outcomes......................................................12
1. INTRODUCTION AND BACKGROUND

In the most general terms, a testbed refers to an integrated set of controlled environments used for experimentation and evaluation. A testbed is meant to serve multiple purposes. For example, it provides users a framework to carry out their experiments, reduces duplication of work to a great extent, maximizes use of invested resources etc. Although the term “testbed” applies to a wide range of disciplines, the focus here is to expand its importance and essence in the area of wireless sensor networking.

1.1 Background

Over the years, usage of mobile devices has spread so much that it subsequently lead to miniaturization of embedded systems. With this, computing devices became even cheaper, smaller and more pervasive in everyday life. This resulted in a desire for smarter environments that could be controlled from anywhere and at anytime. These advancements in hardware technology have propelled the existence of Wireless Sensor Network (WSN). A WSN consists of simple, low cost yet powerful sensors. Each sensor has the ability to sense, process, and communicate data collected from the environment, in which it is deployed. Sensors usually draw energy from a small battery, and thus energy efficiency emerges as the key issue in any WSN. The basic idea of a WSN is to employ a large number of sensors to collectively monitor and disseminate information about a phenomenon of interest. WSNs have been designed to support a diverse range of applications. Some examples include military surveillance, habitat and weather monitoring, agricultural crop management, wildlife monitoring, target tracking,
emergency rescue operations [Akyildiz 2002], and biosensors for health monitoring. It is believed that WSNs will drastically change our lives [Su 2008] in the future. The typical architecture of a wireless sensor node contains a sensing unit, a processing unit, a transceiver unit, a power unit, and an optional mobilizer, location finding system. The processing unit may contain a small memory unit.

In addition to sensor nodes, the network may also contain a sink or base station. A sink is a node with relatively powerful communication and computation ability. It generally serves as a gateway in the network. Different kinds of communication patterns are possible between the sink and sensor nodes [Demirkol 2006]. However, the most common type of communication is convergecast, in which, a group of sensors communicate to a sink. Sensors with networking capabilities are now being made available for commercial and research purposes. Crossbow [Crossbow 2010] is one such leading supplier of wireless sensor technology. Crossbow takes advantage of the Microeletromechanical Systems (MEMS) technology to develop sensor motes. These motes have the ability to communicate with each other to form a wireless sensor mesh. Crossbow offers distinct sensor kits depending upon the usage domain. The most recent sensor system called eKo Pro Series [Crossbow 2010] has been designed specifically for monitoring Health, Environmental and Agricultural systems.

1.2 Simulation

In order to test a WSN, researchers generally use a technique called simulation. Simulation, in the context of WSN, refers to prototyping a physical environment and its functionality. TOSSIM [Levis 2003] is the most widely known simulator for TinyOS
applications that models the abstractions of the TinyOS system by making some simple assumptions. Users can compile their TinyOS applications for the TOSSIM framework instead of compiling it for a hardware mote. TOSSIM provides abstractions for radio, power, energy consumption and more. Additionally, it provides a set of development tools that can be used to manipulate those abstractions. However, the results from TOSSIM are not reliable because of its constantly changing behavior. For this reason, TOSSIM is considered more suitable for evaluating and comparing algorithms.

Network Simulator (NS) [Network 2010] is an open source simulator for wired as well as wireless networks. It provides simulation for mostly routing and multicast protocols. NS is written in C++ and its simulation interface is based on OTcl [Object 2010] scripting language. NS has simulation models for energy constrained wireless networks and therefore allows close prototyping of physical real world networks. Unlike TOSSIM which provides bit level abstraction, NS provides packet level abstraction. One major limitation of NS is that it does not perform well for large scaled network simulation.

1.3 Project Motivation

Simulations can be modeled much more easily when compared to the complex procedure of setting up a physical environment. It is also easy to duplicate and manage simulations. However simulation fails to capture some essential details such as communication losses and delays, resource constraints, etc. The results from simulation are therefore not as accurate and reliable as the results from a realistic environment. This emphasizes the need for a real-time experimental environment that requires one to
manually program the sensor nodes and deploy them in the target environment. However, such tasks are generally complicated and time taking. To address this complexity and enable the development of applications in a more flexible manner, a facility is required to fit the slot between simulation and realistic environment. WSN testbed is a solution that provides a controlled environment for application deployment and testing, apart from numerous other services like creation, scheduling and execution of experiments, automation of node reprogramming and data logging etc. The main idea is to reduce the time and human effort required to carry out real time experiments.

There are several testbeds available for WSNs [Girod 2004] [Emulab 2009] [Werner-Allen 2005]. Most of the testbeds have been developed at educational institutions for teaching and research purposes. Although these testbeds are being widely used, they have some limitations. The reasons for proposing yet another WSN testbed are listed below:

- Most of the existing testbeds do not provide a proper user interface through which users can schedule their experiments and view results. Users are required to possess technical knowledge of scripts in order to be able to access the testbed. Having a Web interface will allow users to easily access the testbed.

- Existing testbeds are either unavailable or impose usage quotas for external users. Although this ensures fair sharing, it might not always be a convenient option for an external user. Currently many WSN researchers in our lab carry out the redundant work of setting up a WSN from scratch. Having our own testbed will reduce duplicate and time consuming tasks to a great extent and also allows sharing of expensive resources.
A testbed within reach can also serve as an effective tool for teaching. Instructors can use the testbed to demonstrate its hardware and software design and operation.

1.4 Related Work

There are many testbeds being developed all across the world. Most of these have been developed at Universities to aid research in networking. The testbeds reviewed by us in the process of understanding and identifying the project requirements, goals, and functions include EmStar, Emulab, Motelab, SenseLAB and PlanetLab. These are discussed below.

PlanetLab [Planetlab 2010] is a constantly improving global network tool being used by a large number of researchers for realistic network experimentations. Using PlanetLab, one can conduct experiments on a large number of widely distributed machines across the world. PlanetLab has several servers in different parts of the world each of which hosts several programs supporting a myriad of applications. PlanetLab is comprised of several architectural elements described below:

- Site: Physical location of a PlanetLab node.
- Node: Server dedicated for running PlanetLab services.
- Slice: Set of resources allocated on distributed PlanetLab nodes.
- Node Manager: A program running on the node responsible for creating slices on the node and also for managing resources.
- Auditing Service: It records information about packets sent from a node.
- Slice Authority: Stores the state of a set of system wide slices.
Management Authority: Performs software installations and updates on nodes.

When an organization decides to be a part of PlanetLab, two of the organization’s nodes are dedicated to run PlanetLab services. In exchange, the organization is allowed access, to a small slice on each node of PlanetLab, through a shell script. Users of the organization will then be able to develop and deploy their own network applications on PlanetLab. Some commonly provided services on PlanetLab include the following:

- **CoDeen**: It is a proxy server that allows users to set their Internet cache to nearby high bandwidth proxy. This service was designed to achieve Content Distribution on PlanetLab.

- **CoMon**: It supplies monitoring statistics for nodes on PlanetLab. Users are provided commands to query the status and errors on nodes.

- **ScriptRoute**: It is a tool designed to detect traffic anomalies and examine errors.

Although PlanetLab is the most widely used platform for conducting network experiments, it has several limitations [Spring 2006]. Firstly, because a large number of users access the nodes, the nodes can fail and become unavailable making it difficult to repeat experiments. Secondly, though PlanetLab is hyped to be an overlay network on top of the Internet, many studies reveal that PlanetLab is not a true representation of the Internet. Finally, PlanetLab does not guarantee use of resources because it depends on the availability of resources.

Motelab [Werner-Allen 2005] is a testbed specifically designed for wireless sensor networks. It was designed at Harvard University and is widely used for both research and teaching purposes. It comprises of approximately 30 Crossbow Micaz motes connected
via an Ethernet. It also has 20 MIB interface boards that communicate with the Micaz motes. Motelab has a Linux server with Apache, MySql and PHP Web interfaces. Users access Motelab through the Web interfaces. The main architectural components of Motelab include the following:

- **Motelab Hardware Components**: These include MicaZ motes, MIB 600 interfaces, Ethernet cables, etc.
- **Motelab Job**: It is a mapping that describes how each uploaded executable should be mapped to Micaz motes.
- **MySql Database**: It has a set of databases holding information such as state of the testbed nodes, user information, uploaded files, jobs created and results from running the jobs.
- **Web Interfaces**: PHP dynamic Web pages that allow authorized users to login and gain access to perform experiments on the testbed.
- **DBLogger**: it is a java program that runs on each sensor node. It collects data from all sensors and translates it into useful information that is stored in the database.
- **Job Daemon**: it is a Perl script that handles reprogramming of nodes, killing running jobs and providing properly formatted log data for the users to download.

Motelab also provides a user quota system that enforces resource sharing among multiple users. Each user is allowed a time of 30 minutes per day to run their tasks. Users are also permitted to directly access the serial port of nodes through a TCP/IP connection. Like Planetlab, Motelab also has its limitations. The usage quotas may ensure fair access
to resources but cause lot of inconvenience to external users who wish to perform lengthy experiments.

EmStar [Girod 2004] provides a platform for developing WSNs. EmStar offers a set of tools that can be used for deployment, simulation, emulation, and visualization of real as well as simulated systems. EmStar provides services like networking, sensing and time synchronization. One key feature missing in EmStar is the ability to conduct system wide compile time optimizations. EmStar provides three modes for WSN experiments: pure simulation, real time experiments and hybrid mode. For the real time experiments, EmStar uses crossbow motes on the ceiling of the lab. Hybrid mode has the advantage that it provides powerful visualization tools. It uses two main tools to be able to work in different modes – EmSim and EmCee.

- EmSim is a simulation environment that allows several virtual nodes to run parallely while communicating with simulated modules and radio channels.
- EmCee is a variant of EmSim with an interface to use real radio transceivers to interact.

Emulab [Emulab 2010] is the first to provide a mobile wireless sensor network testbed that allows its users to develop, diagnose and assess their systems. It is open to public free of cost and has two primary goals: scalability and flexibility. Emulab consists of hundreds of test nodes, multiple servers (master, database, Web and DNS), routers, Ethernet ports, and power controllers. Emulab attempts to hide the heterogeneity of its resources. Emulab provides several different environments for experimentation:

- Simulation: Network Simulator (NS) is used to define topologies
- Live Internet Experimentation: An interface is provided to PlanetLab to allow users to build and test their applications across the world.
- 802.11 Wireless: 802.11 Wifi environment for experimentation.
- Emulation: Topology allowing full control of nodes.

SenseLAB [Sense 2010] was developed by INRIA very recently, one of the leading research centers in France. The goal was to serve as an efficient research tool for developing, deploying and testing WSN applications. Apart from providing a platform for experimentation, SenseLAB also provides a set of complex tools using which users can also simulate and emulate sensor applications.

A comparison of testbeds discussed so far is presented in Table 1.1 below.

<table>
<thead>
<tr>
<th>Testbed</th>
<th>Wired/Wireless</th>
<th>Scale</th>
<th>Nodes</th>
<th>Area</th>
<th>Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>PlanetLab</td>
<td>Wired</td>
<td>1090</td>
<td>Servers</td>
<td>Distributed</td>
<td>Consortium of institutions</td>
</tr>
<tr>
<td>Motelab</td>
<td>Wireless</td>
<td>190</td>
<td>Micaz, Telos, MIB600</td>
<td>Indoor</td>
<td>Harvard University</td>
</tr>
<tr>
<td>Emstar</td>
<td>Wireless</td>
<td>70</td>
<td>Mica1 &amp; Mica2</td>
<td>Indoor</td>
<td>UCLA</td>
</tr>
<tr>
<td>Emulab</td>
<td>Wired/Wireless/Mobile</td>
<td>500</td>
<td>PC nodes, 802.11x nodes</td>
<td>Indoor</td>
<td>University of Utah</td>
</tr>
<tr>
<td>SenseLAB</td>
<td>Wired/Wireless</td>
<td>1024</td>
<td></td>
<td>Distributed</td>
<td>INRIA</td>
</tr>
</tbody>
</table>

Table 1.1 Comparison of Testbeds
1.5 Contributions of the Project

At the Science and Technology Department, we built a small experimental testbed for WSNs called Scibed. My contributions towards the project are as follows:

1. Developed user-friendly Web interfaces for the users to gain access to the testbed.
2. Designed a setup for the testbed allowing users to perform experiments on the testbed.
3. Implemented existing protocols and applications to enable the proper functioning of testbed.
2. WIRELESS SENSOR NETWORK TESTBED

2.1 Project Scope

WSNs are evolving towards the most important technologies for the future Internet. Their purpose is to monitor an environment to analyze process and deliver the collected data to interested parties.

Until date, there have been limited numbers of WSN testbeds that allow full access to their testbeds. The goal of this project was to design an experimental wireless sensor network testbed that would be accessible to researchers from the Science and Technology department of our University. The idea was to enable researchers to get around the time consuming and redundant task of designing a wireless sensor network. Users who have been authorized will have access to the experimental facilities of the testbed. Using the testbed users can run their applications and verify the results. A friendly and easy to use Web-interface has been provided to simplify access to the testbed. The interface will allow users to request accounts, login, upload programs, and query data. Additionally, a separate Web Interface for administrators has been provided to create, manage, and delete accounts.

The appendix provides useful information on setting up a small testbed and scaling it to a bigger size. This includes information on how to install various software packages and wire the components together.

2.2 Project Objectives

The objectives of this project were to:
- Study and thoroughly analyze the goals, design and functioning of existing testbeds.
- Identify services that will be made available via the testbed.
- Develop user-friendly Web interfaces for the users to gain access to the testbed.
- Develop user-friendly Web interfaces for the administrator to manage users.
- Design a setup for the testbed allowing users to perform experiments on the testbed.
- Implement existing protocols and applications to enable the proper functioning of testbed.

2.3 Project Strategy

The following strategy has been adopted for developing the project.

- **Analysis:** Conducted a thorough analysis of what we intend to accomplish and derived a requirements list from the analysis.
- **Design:** Here we mainly focused on deriving a system design from the requirements gathered previously.
- **Development:** This involved the actual development of the system i.e., the testbed and other associated tasks like creating Web interfaces.
- **Evaluation:** Reviewed and evaluated the system based on its functioning.
- **Support:** Provided support in the form of documentation to users of the testbed.

Future work will include having a manual and FAQs and other support documentation on the Web page.
2.4 Project Stages and Outcomes

This section describes a systematic approach that has been followed to fulfill the project objectives. The main idea of adopting such an approach was to ensure that the project is conducted in a consistent manner. Table 2.1 below lists the stages and actual outcomes in each phase.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1. Analysis</strong></td>
<td><strong>Specifications Document</strong></td>
</tr>
<tr>
<td><em>Kick off meetings to determine:</em></td>
<td></td>
</tr>
<tr>
<td>a. Purpose</td>
<td></td>
</tr>
<tr>
<td>b. Project requirements</td>
<td></td>
</tr>
<tr>
<td>c. Comprehensive list of resources</td>
<td></td>
</tr>
<tr>
<td>d. Project Schedule</td>
<td></td>
</tr>
<tr>
<td><strong>2. Design</strong></td>
<td><strong>System architecture – hardware, and software</strong></td>
</tr>
<tr>
<td><em>Design the following:</em></td>
<td></td>
</tr>
<tr>
<td>a. Overall architecture of the system</td>
<td></td>
</tr>
<tr>
<td>b. User Interface</td>
<td></td>
</tr>
<tr>
<td><strong>3. Implementation</strong></td>
<td><strong>Web page and testbed</strong></td>
</tr>
<tr>
<td><em>Perform the following tasks:</em></td>
<td></td>
</tr>
<tr>
<td>a. Install &amp; configure hardware and software</td>
<td></td>
</tr>
<tr>
<td>b. Design Web interfaces</td>
<td></td>
</tr>
<tr>
<td>c. Develop testbed</td>
<td></td>
</tr>
<tr>
<td>d. Demonstrate testbed</td>
<td></td>
</tr>
<tr>
<td><strong>4. Evaluation</strong></td>
<td><strong>Revised report with scope for improvement</strong></td>
</tr>
<tr>
<td><em>Review Sessions</em></td>
<td></td>
</tr>
<tr>
<td><strong>5. Support</strong></td>
<td><strong>FAQs, User Manuals etc</strong></td>
</tr>
<tr>
<td><em>Provide continual support to users of the testbed</em></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1 Project Stages and Outcomes

2.5 Project Requirements

Requirements for this project are a set of essential characteristics that we drafted for the system to function according to the specifications. Two types of requirements namely
functional and non-functional requirements are listed below. While functional requirements describe how a specific component must serve, non-functional requirements specify how the system must behave.

2.5.1 Functional Requirements

- **Physical Environment Modeling**: The system should closely model the physical environment.
- **Reprogramming Ability**: The system should allow users to reprogram the sensor nodes.
- **Power-aware**: The system should take into account energy and memory constraints of sensor nodes.
- **Web Interface**: The system should provide a user friendly Web interface for accessing the testbed.
- **Database**: The system should include a database, which stores user data and results of experiments.

2.5.2 Non-functional Requirements

- **Availability**: The system should be available 24/7
- **Reliability**: The system should be trustworthy and must produce accurate results.
- ** Recoverability**: The system should be able to recover from any failures.
- **Security**: The system should not allow access to unauthorized users and should be protected against attacks.
• Data Integrity: The data stored and transmitted must be complete and correct.

• Efficiency: The system should have a good performance.

• Usability: Users should find the system easy to use.

• Portability: The system should be designed in a way such that it is easily portable to other operating systems.

2.6 System Description

The main outcome of this project is the Web-based testbed consisting of wireless sensor nodes. The testbed allows authorized users to perform wireless sensor experiments on it. Additionally, the testbed will also serve as an effective tool for teachers as well as researchers. The system contains a set of components as discussed below. Figure 2.1 shows the flow of events among these components.

2.6.1 Users

Any user will need an account to access the testbed. Users will be allowed to request accounts via the Web Interface. The administrator reviews information submitted by the user and decides whether or not to approve the request. Services of the testbed can be accessed once the user logs into his account. An authorization system is implemented to ensure that only genuine users have access to the testbed. Each user will be able to upload his tasks such as running programs on the testbed and reading results of the program execution.
2.6.2 Web Interface

Users are presented with a User Interface through which they will be able to access the services of the testbed. Users can request accounts, log into their accounts, create jobs/tasks, and query data through the interface. Administrators of the testbed will be presented with a similar interface through which they can manage accounts, and control testbed functionality.

2.6.3 Database

The database will store data collected from the experiments, information about users and administrators, information used to display the Web pages and state of the testbed operation.

2.6.4 Core System

The core system consists of the main software and hardware components of the testbed. It also includes a group of sensor nodes, which will perform the user requests and return the collected data.

Figure 2.1 Core System Components
3. SYSTEM DESIGN

The previous sections discussed several existing solutions for WSN testbed. However, most of the testbeds discussed above do not give full access to an external user. Therefore, a new testbed design was presented with reference to existing solutions. This Section explains how the system design has been realized. The content is organized as follows: Section 3.1 describes major components of the testbed system. Section 3.2 discusses the actual architecture of the testbed. Lastly, Section 3.3 presents a comparison of our testbed solution to existing solutions.

3.1 Components of Proposed System

The Web-based testbed includes a set of sensor nodes, a base station that connects the wireless mesh to a data repository and a Web server. These components are discussed below.

3.1.1 Hardware

We use Crossbow’s MICAz [Crossbow 2010] sensor motes to form the wireless mesh. The MICAz is a 2.4 GHz module used to form a low power WSN. Figure 3.1 shows the picture of MICAz. The mote complies with IEEE 802.15.4 standard [Woon 2007]. It has an Atmega 128L microcontroller with 4 Kbytes SRAM and 128 Kbytes of flash memory. The mote has built-in temperature, pressure, acceleration, barometric, acoustic, and magnetic sensing abilities.
Any MICAz node can operate as base station when attached to the MIB520 USB interface board. The MIB520 provides power to the devices via a USB bus. MIB520 has an on-board processor, Atmega 16L, which it uses to receive programs from the USB port and later programs the motes. The MIB520 provides two separate ports – one for in-system mote programming and another for data communication over USB. A picture of MIB520 is shown in Figure 3.2.
3.1.2 Operating System

Tiny OS [Gay 2005] is an open source development environment specifically designed for wireless embedded networks. It supports MICAz and several other wireless modules. The main objective behind designing Tiny OS is saving energy. Therefore, it is ideal for embedded systems with scarce resources. Tiny OS is written in nesC language [Gay 2003], an extension to C with additional features like components and concurrency. All nesC applications comprise of one or more components. These components are combined together to form an executable. Each component has an interface specification. Interfaces have a set of commands and events. Commands are written by the interface provider and events are written by the interface user. Components are of two types namely module and configuration. Module comprises the actual implementation of interfaces and configuration wires all components together. For our Web-based testbed, we used Tiny OS 2.x for the operating system.

3.1.3 Sense: NesC Example Application

Here we describe the components, function, compilation and installation of a NesC application called Sense [TOS 2010]. This application collects sensor readings at regular intervals and outputs the values on the LEDs. The main components of Sense include a configuration file called “SenseAppC.nc”, a module file called “SenseC.nc”, and a make file. The nesC compiler reads the actual source file, SenseAppC, and creates an executable from it. SenseC is the program that provides the implementation of Sense application. SenseAppC is responsible for wiring SenseC and any other required components. All TinyOS applications have a make file that triggers the nesC compiler
and also allows users to choose their hardware platform for compilation. To compile Sense application for micaz, go to `/tinyos-2.x/apps/Sense/` and type `make install` as shown in Figure 3.3. This will create a `tos_image.xml` file that can be run on a micaz mote.

```
make micaz install,1 mib520,/dev/ttyUSB0
```

Figure 3.3  Output of Compilation

To download the program on to the mote, connect a mote on the programming board and connect the programming board to the computer via a USB bus. This will turn on the green LED on the programming board indicating proper power supply. Type the following command on the terminal to download the program on to the mote and a successful installation will give the output shown in Figure 3.4.

```
make micaz install,1 mib520,/dev/ttyUSB0
```

We now explain the above command. Make is a powerful file that invokes the compiler and provides several options for compilation. Micaz is the platform for which make is
compiling the code. If the application has to be compiled for TOSSIM, the command will be `make mica2 sim`. The number 1 indicates the identity assigned to the mote being programmed. Mib520 is the interface board and `/dev/ttyUSB0` is the port device name.

Like it was mentioned above, MIB520 provides two ports for programming and communicating with the motes. To find out what ports are being used, type the following command: `ls -la /dev/ttyUSB*`.

---

**Figure 3.4 Output of Program Installation**

3.1.4 Deluge T2 – Data Dissemination Protocol

Deluge [Hui 2005] is a part of Tiny OS 2.x systems and serves as a reliable data dissemination protocol. It propagates program images to sensor motes, which then reset with that new program image. It has a communicable way of spreading the images. Sensor nodes periodically broadcast advertisements. These advertisements contain a
version number and vector to indicate the version of the program available on it. When a
sensor node realizes that it has to update a program from an advertisement, it sends a
request to the advertiser for the required packets. Once a sensor node updates its program,
it starts to broadcast advertisements to its neighboring nodes. This will enable all sensor
nodes to possess the same version of program. Wrapper is an addition to Deluge, which
sends an alert to the user once the node obtains the new program image. Additionally,
users will be able to turn off the propagation feature of Deluge. This makes it possible to
have different versions of programs on different nodes. Deluge has the following
features:

- Multihop support – wireless programming of nodes in a multihop network.
- Epidemic propagation – continuous propagation of program images.
- Redundant data integrity checks – checksum calculations to ensure integrity of
  program images.
- Multiple program images – enable switching between program images without
  continuous downloading.
- Golden image – a block of code for recovery purposes.
- Isolated bootloader – takes care of programming the microcontroller.
- RAM footprint – provides storage for less than 150 bytes.

In the example below, we show the procedure for disseminating a program from the
base station to the remaining nodes in the network.

- The first step is compile a “tosboot” application which is a bootloader for deluge
  T2. To compile the tosboot, navigate to /tinyos-2.x/tos/lib/tosboot and run the
  command make micaz.
The second step is to compile and install “BaseStation” application on one of the motes, which will then act as the base station for the rest of this exercise. To compile and install this application, connect a mote to the programming board and the board has to be connected to the computer via a USB bus. Then, navigate to /tinyos-2.x/apps/tests/deluge/BaseStation and run the compile and install command: make micaz install,0 mib520,/dev/ttyUSB0. The output should be as shown in Figure 3.5. The BaseStation application has no function other than initializing the deluge component on the base station.

```
peetha@peetha-laptop:/local/src/tinyos-2.x/apps/tests/deluge/BaseStation$ make micaz reinstall,0 mib520,/dev/ttyUSB0
```

![Figure 3.5 Preparing the Base Station](image)

The third step is to initialize the deluge component on the client motes. To do so, we install the “GoldenImage” application on all the motes in the network.

Connect the mote to the programming board and while in the terminal, navigate to
The fourth step involves preparing the application that needs to be disseminated. There are three things to be verified in the application:

a. The main application file should include the Deluge component i.e., it must include the following statement: `Components DelugeC`

b. The make file should include a statement to initialize the bootloader:

```
BOOTLOADER = tosboot
```
c. There should be a “volumes-at45db.xml” file in the application folder. This file defines the storage volumes by name and size in bytes.

- Once the application is ready, the next step requires injecting the application into the base station. The command used for this is `tos-deluge`. To download the image file to the base station, run the following command:

```
tos-deluge serial@/dev/ttyUSB1:57600 –i 1 apps/.../build/micaz/tos_image.xml
```

where “…” is the application folder. This command injects the program image file into the base station on the flash volume labeled 1. The output of this command is shown in Figure 3.7.

![Figure 3.7 Injecting Image File Into Base Station](image)

- The final step is to disseminate the file on the base station to the remaining nodes. This is using the `tos-deluge` command again:
tos-deluge serial@/dev/ttyUSB1:57600 -d 1

This command informs the base station to advise all the nodes in the network about the availability of a new program image. Figure 3.8 shows the base station sending the command to all active nodes.

Figure 3.8 Dissemination Command Sent to Network

Deluge offers a simple command line utility [TOS 2010] that allows a user to perform multiple queries.

- `-p`: this option provides information about the images currently executing and installed on a particular mote. The information given includes program name, user ID, Compiled date and time, hardware platform, host name, user hash, size and number of pages.
-i: this option takes the tos_image.xml file as input and creates and injects an image file into the specified flash volume of the mote.

- d: this option informs the base station to notify the notes in the network about the availability of a new program image.

-dr: this option is for both dissemination and running the image file and forces a reboot.

-e: this option is used to erase a volume on the flash memory of the base station.

-s: this option stops the propagation of image file any further.

-ls: this option is used to unlock a volume to perform any operations on it.

### 3.1.5 Serial Forwarder – Data Collection Component

Serial forwarder [TOS 2010] is a java program that is used to collect data from the network. Additionally, it allows users to send commands to the motes in the network and also to keep a track of the network traffic.

- The first step in setting up the serial forwarder is to check the communication between the PC and the mote by installing ‘TestSerial’ application. This communication is achieved through serial port. Figure 3.9 shows the output of successful installation of testserial.

- In the next step, run the associated java application that communicates with the serial port. To do so, run the following command: `java TestSerial –comm serial@/dev/ttyUSB1:57600`. This command instructs the application to listen to the serial port. The output of this is shown in Figure 3.9.

- Finally, we use the listen tool to print out the packets that it sees. This is shown in
Figure 3.10. The command to use Listen is: `Java net.tinyos.tools.Listen`

Figure 3.9  Installation of TestSerial Application

Figure 3.10 Displaying Packets Sent and Received
3.1.6 Software Framework

The Web-based testbed comprises of three main modules. Figure 3.2 shows an organization of the modules. These modules are available to the user via the Web interface, some directly and some indirectly.

- **Uploading module** – This module uploads programs to sensor nodes.
- **Reading module** – This module is used to read data from sensor nodes.
- **Database module** – This module is used to store the data read from sensor nodes.

In the uploading module, the base station sends instructions to every other sensor node in the network and ensures that nodes have received instructions. Failures are reported if nodes do not function as expected. Dissemination of information can be of two types: one-to-one mode in which commands are propagated from the base station to a
fixed destination node and one-to-all mode in which commands are propagated to all nodes in the network. Our testbed uses one-to-all mode of transmission.

Feedback is a method to confirm successful delivery of the message, and collect useful information from nodes. Feedback can be active or passive. Passive feedback is activated when requested whereas in active feedback, feedback will be sent automatically on the occurrence of an event. Feedbacks can be classified into four types based on their functions:

- Command dissemination feedback – to confirm that commands have been received by destination node.
- Data collection feedback – to confirm the start of collection of information.
- Error reporting feedback – to report different errors to users.
- Confirmation feedback – to confirm which nodes have received a new image.

Scibed achieves feedback using Deluge and its options.

Figure 3.12 Testbed Modules
Data will be generated in most of the experiments performed on the testbed. This data is very useful for analyzing the behavior of network. The logged data is usually stored in remote sensor flash memory and is collected on completion of the user program. Data logging and data collection are therefore two main functions of the testbed.

3.1.7 Graphical User Interface (GUI) Tools

A friendly user interface is provided to the user to access the services of the testbed. The GUI software is written using PHP. The GUI for users is different from the GUI for administrators of the testbed. The GUI for users has the following services:

- General information about the testbed.
- Registration for new users.
- Login option for existing users.
- Modifying account information.
- Schedule option for users to schedule their tasks.
- Upload option for users to upload their programs.
- Results option for users to view results of their programs.

The GUI for administrator has the following services in addition to the services listed above:

- Creation of user accounts.
- Managing user accounts
- Deletion of user accounts.
- Approving user jobs
- Updating FAQs and other Web content.

Web interfaces for the system are shown in Figures 3.3 through 3.9. The Index page is similar to home pages of other Websites. It provides general information about the testbed and provides options for users to request and access their accounts. The Setup page shows how the testbed is organized. In the Request Account page, users can request an account with the testbed. The information will be submitted to the Administrator who will decide to approve or reject the request. The Login page allows authorized users to sign into their accounts. Users can schedule jobs using the Schedule Job page. Manage Jobs page will allow the user to manage the scheduled jobs. The Contact page provides contact information of the administrator. Similar interfaces are provided to the administrator for creating, managing, deleting accounts and running tasks.
**TestBed Setup**

SciBed is an open source WSN testbed consisting of Arduino wireless nodes and WEBS212 Gateway. The operating system is Linux, and the backend is a MySQL database. Apache Webserver, MySQL database store the information about sensors and results.

**Figure 3.14 Setup Page**

**Request Account**

The user enters the name, last name, email address, and academic institution. The user submits the form to request an account.

**Figure 3.15 Request Account Page**
Figure 3.16 Login Page

Figure 3.17 Schedule Job Page
Figure 3.18 Manage Jobs Page

Figure 3.19 Contact Page
Figure 3.20 Create Users Page

Figure 3.21 Delete Users Page
Figure 3.22 Update User Page

Figure 3.23 Manage Jobs Page
3.2 Testbed Architecture

Figure 3.8 shows the architecture of our testbed. Client is a user who wishes to perform experiments on the testbed. Authorized users can access the testbed via a Web-based user interface. Users will be able to schedule tasks, upload programs and read results. Tiny OS is built on the host operating system and will communicate between its components and the base station. When a user uploads a program, Tiny OS will make use of Deluge to propagate the program image to the base station and other sensor nodes. When all the nodes contain the updated program image, an alert is sent and the sensors start gathering the information for which they have been programmed. The base station aggregates the data gathered from the sensors and invokes the serial forwarder, which forwards the data to a serial port. This data is then populated into the database. When a user requests to read the data from an experiment, the data is made available to the user from the database.
### 3.3 Discussion

The objective behind this project was to design and implement a small, reliable, low-cost testbed accessible to authorized users. We adopted some good characteristics from existing testbeds and added more enhanced features to it. Most of the WSN testbeds discussed in the previous section use additional hardware like Ethernet interface boards and micro servers. These devices are used to enhance the performance and to make the testbed more powerful. For example, micro servers function as more powerful nodes. They have larger memory and faster processing capacity and so can handle more work. Therefore, introducing micro servers will decrease the load on the main server. Apparently, the reliability of these services is much higher than the ones without additional devices. However, all these devices are dependent on Ethernet connection. Although, it is easy to setup an Ethernet connection in a small area, it becomes difficult to manage the Ethernet with the size of testbeds. The challenge we confront is reliability of the WSNs without the additional devices.
4. EVALUATION AND RESULTS

4.1 Evaluation

Upon completion of the project, the following tests were performed to evaluate the system and ensure that it functions accurately.

1. Compatibility of the user interface with browsers such as Internet Explorer, and Mozilla Firefox.
2. Usability of the interface.
3. Registration of new users.
4. Authentication of valid users.
5. Failed authentication of invalid users.
6. Ability of users to upload their programs through interface.
7. Dissemination of programs to the sensor motes.
8. Ability of motes to sense data precisely.
9. Populating collected data from sensors into the database.
10. Displaying data from the database.
11. Integration among individual software modules.
12. Functioning of the system as a whole.

4.2 Results

A testbed application with fully functional graphical user interfaces and administrator interfaces was developed. We were able to implement and have demonstrated (in sections 3.1.4 through 3.1.7) the successful dissemination and data collection to and from the nodes in the network. However, we are facing some
challenges while trying to perform repeated experimentations. Some motes are reluctant to accept programming and so manual intervention is now required in running the experiments. In trying to identify the source of this problem, we came across few other users who were experiencing the same problem with micaZ motes.
5. FUTURE WORK

There are three main components of the testbed: Web Interface component, Data Dissemination component and Data Collection component. Though each component is delivering its desired functionality, there is a need to integrate these three components into a single unit to make the system completely functional. A PHP script is required to automate testbed functions which include program uploading and querying results. Detailed information on how this integration can be achieved is given in Appendix C.

Much more effort has to be put in order to scale the existing testbed system to a larger number. Scibed is an experimental testbed with as few as 6 motes. The current setup can support only upto 20 motes. Also, it can support very few users without degrading performance of the testbed. This becomes a limitation when a large number of users wish to access the testbed simultaneously.

Serial forwarder is the tool used in Scibed for reading data from the serial port. Though it presents a benefit of allowing more than one application to connect to it, TinyDB offers a easier and featured way of querying the data. Currently, TinyDB is not available for TinyOS-2.x but it is expected to be made available soon. A future addition to the testbed would be to use the TinyDB querying system to read data from the serial port.

Scibed has been designed only for internal use. In the future, we would like to make our testbed accessible to remote users allowing them to contribute to research in Wireless Sensor Networks.
6. CONCLUSION

We studied several existing WSN testbeds. However, many of these existing testbeds impose usage quotas making it an inconvenient option to external users. We built an experimental testbed on which internal researchers can test their applications. The testbed solution is based on existing solutions and is modified to suit a new set of requirements. The testbed comprises of Crossbow’s MICAz motes and several software modules offering services like command dissemination and data logging. Authorized users will be able to access these services via a Web interface. In order to ensure that everything works okay, we used several test cases for testing the different modules of the system.


APPENDIX A. PHP SCRIPTS AND PROGRAMS

// ***********************************************
// Script Name: insert_request.php
// Task: This script uses phpmailer class to submit a request from the user to the
// administrator via SMTP.
// Reference for phpmailer class: http://phpmailer.sourceforge.net

<?php
    include_once("connection.php");
    $fname=$_POST['fname'];
    $lname=$_POST['lname'];
    $email=$_POST['email'];
    $institute=$_POST['institute'];
    $url=$_POST['url'];
    $isstudent=$_POST['isstudent'];
    if($isstudent!=1)
        $isstudent=0;
    $ad_name=$_POST['ad_name'];
    $ad_email=$_POST['ad_email'];
    $intend=$_POST['intend'];
    $query="INSERT INTO `scibed`.`requests` (`fname`, `lname`, `email`, `institute`,
    `url`, `isstudent`, `ad_name`, `ad_email`, `intend`) VALUES ('$fname', '$lname', '$email',
    '$institute', '$url', '$isstudent', '$ad_name', '$ad_email', '$intend')
    
    mysql_query($query);

    require("class.phpmailer.php");
    $mail = new PHPMailer();
    $mail->IsSMTP(); // Send through SMTP
    $mail->SMTPAuth = true; // SET SMTP authentication
    $webmaster_email = 
    "geetha.sanapala@gmail.com"
    $email2="geetha.sanapala@gmail.com"
    $name="Geetha Sanapala"
    $mail->From = $webmaster_email;
    $mail->FromName = "SCIBED new Request"
    $mail->AddAddress($email2,$name);
    $mail->AddReplyTo($webmaster_email,"Webmaster")
    $mail->WordWrap = 100; // Word wrap
    $mail->IsHTML(true); // Send as HTML
    $mail->Subject = 'New Request from '.$fname.' '.$lname;
    $mail->Body = 'Confirm the following information:<br/>User email :
    '.$email.'<br/>Institute: '.$institute.'<br/>Url : '.$url.'<br/> Advisor name :
    '.$ad_name.'<br/>Advisor email : '.$ad_email.'<br/>Intended :'.$intend;
    $mail->AltBody = "You are viewing plain text"; // Text body
    $mail->Send();
header("Location:thankyou.php");
?>

//

******************************************************************************
******
// Script Name: action_login.php
// Task: User authorization
<?php
include_once("connection.php");
$user_name=$_POST['user_name'];
$password=$_POST['password'];
$query = "select * from users where user_name='$user_name' and password='$password';
$result = mysql_query($query);
$row = mysql_fetch_row($result) ;
// echo $query;
if($row)
{
    session_start();
    $_SESSION['user']=$user_name;
    header("Location:manage.php");
}
else
    header("Location:login_error.php");
?>

//

******************************************************************************
******
// Script Name: action_schedule.php
// Task: Submits the inputs into the users respective database.
<?php
include_once("connection.php");
if(isset($_POST['Submit']) && $_FILES['userfile']['size'] > 0)
{
    $fileName = $_FILES['userfile']['name'];
    $tmpName = $_FILES['userfile']['tmp_name'];
    $fileSize = $_FILES['userfile']['size'];
    $fileType = $_FILES['userfile']['type'];
    $fp= fopen($tmpName, 'r');
    $content = fread($fp, filesize($tmpName));
$content = addslashes($content);
fclose($fp);

session_start();
$user=$_SESSION['user'];
$date=$_POST['date'];
$time=$_POST['time'];
$query="INSERT INTO `scibed`.`$user` (`date`, `time`, `file`, `results`) VALUES
  ('$date', '$time', '$fileName', '');
mysql_query($query);

header("Location:manage.php");
}

<?php
//
// ********************* ********************* ******************** **********
******
// Script Name: connection.php
// Task: Connects to the specified database

$dbhost = 'localhost';
$dbuser = 'root';
$dbpass = '*****';

$conn = mysql_connect($dbhost, $dbuser, $dbpass);

$dbname = 'scibed';
mysql_select_db($dbname);
?>

<?php
//
// ********************* ********************* ******************** **********
******
// Script Name: action_adminlogin.php
// Task: Admin authorization

include_once("AdminConnection.php");
$user_name=$_POST['user_name'];
$password=$_POST['password'];
$query = "select * from adminlogin where user_name='$user_name' and password='$password'"
$result = mysql_query($query);
$row = mysql_fetch_row($result);$content = addslashes($content); fclose($fp); session_start(); $user=$_SESSION['user']; $date=$_POST['date']; $time=$_POST['time']; $query="INSERT INTO `scibed`.`$user` (`date`, `time`, `file`, `results`) VALUES ('$date', '$time', '$fileName', ' Sylvester'); mysql_query($query); header("Location:manage.php");
} ?>

//
// ********************* ********************* ******************** **********
******
// Script Name: connection.php
// Task: Connects to the specified database

$dbhost = 'localhost';
$dbuser = 'root';
$dbpass = '*****';

$conn = mysql_connect($dbhost, $dbuser, $dbpass);

$dbname = 'scibed';
mysql_select_db($dbname);
?>

<?php
//
// ********************* ********************* ******************** **********
******
// Script Name: action_adminlogin.php
// Task: Admin authorization

include_once("AdminConnection.php");
$user_name=$_POST['user_name'];
$password=$_POST['password'];
$query = "select * from adminlogin where user_name='$user_name' and password='$password'"
$result = mysql_query($query);
$row = mysql_fetch_row($result) ;

48
if($row)
{
    session_start();
    $_SESSION['admin']=$user_name;

    header("Location:AdminManageUsers.php");
}

else
    header("Location:login_error.php");

//
(fixture1)

// Script Name: action_create.php
// Task: Creates an entry for new user along with a separate database in the login name of the user.

<?php
include_once("connection.php");
$user_name=$_POST['user_name'];
$password=$_POST['password'];
$query = "INSERT INTO `scibed`.`users` (`user_name`, `password`) VALUES ('$user_name', '$password');"
$result = mysql_query($query);  
$query="CREATE TABLE IF NOT EXISTS `$user_name` ( 
`date` varchar(12) NOT NULL,  
`time` varchar(10) NOT NULL,  
`file` blob NOT NULL,  
`results` varchar(111) NOT NULL  
) ENGINE=MyISAM DEFAULT CHARSET=latin1;";
mysql_query($query);
    header("Location:AdminCreate.php");

?>
(fixture1)

//
(fixture1)

// Script Name: action_delete.php
// Task: Deletes entry for the user and drops the table in the login name of the user.

<?php
include_once("connection.php");
$user_name=$_POST['user_name'];
$query = "DELETE FROM `scibed`.`users` WHERE `users`.`user_name` = '$user_name' LIMIT 1";
$result = mysql_query($query);
$query="DROP TABLE `".$user_name."`";
mysql_query($query);
header("Location:AdminDelete.php");

// Script Name: action_manage.php
// Task: Updates the fields as required.

<?php
include_once("connection.php");
$user_name=$_POST['user_name'];
$olduser=$_POST['olduser'];
$password=$_POST['password'];
$query = "UPDATE `scibed`.`users` SET `user_name` = '$user_name', `password` = '$password' WHERE `users`.`user_name` = '$olduser' LIMIT 1";
$result = mysql_query($query);
$query="RENAME TABLE "$olduser" TO "$user_name;";
mysql_query($query);
header("Location:AdminManage.php");

//

PHP WEB INTERFACES

<?
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Home Page</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<link rel="stylesheet" href="mm_travel2.css" type="text/css" />
<script language="JavaScript" type="text/javascript">

//----Declaring variables----
var d=new Date();
var monthname=new Array("January","February","March","April","May","June","July","August","September","October","November","December");
var TODAY = monthname[d.getMonth()] + " " + d.getDate() + ", " + d.getFullYear();
//-----------------------------
<table>
<thead>
<tr>
<th>Request Account</th>
<th>Login</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="mm_spacer.gif" alt="Spacer" /></td>
<td><img src="mm_spacer.gif" alt="Spacer" /></td>
<td><img src="mm_spacer.gif" alt="Spacer" /></td>
</tr>
</tbody>
</table>

```html
<form action="insert_request.php" method="post">
<table border="0" cellspacing="0" cellpadding="0" width="305">
<tr>
<td colspan="3" class="pageName">Request Account</td>
</tr>
<tr>
<td colspan="2" class="bodyText"><p>First Name</p></td>
<td width="166" class="bodyText"><label>
<input type="text" name="fname" />
</label></td>
</tr>
<tr>
<td colspan="2" class="bodyText">Last Name</td>
<td class="bodyText"><input type="text" name="lname" /></td>
</tr>
<tr>
<td colspan="2" class="bodyText">Email</td>
<td class="bodyText"><input type="text" name="email" /></td>
</tr>
<tr>
<td colspan="2" class="bodyText">Academic Institute</td>
<td class="bodyText"><input type="text" name="institute" /></td>
</tr>
</table>
</form>
```
<table>
<thead>
<tr>
<th>Webpage URL</th>
<th>&lt;input type=&quot;text&quot; name=&quot;url&quot; /&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>&lt;input type=&quot;checkbox&quot; name=&quot;isstudent&quot; value=&quot;1&quot; /&gt;</td>
</tr>
<tr>
<td>Advisor Name</td>
<td>&lt;input type=&quot;text&quot; name=&quot;ad_name&quot; /&gt;</td>
</tr>
<tr>
<td>Advisor Email</td>
<td>&lt;input type=&quot;text&quot; name=&quot;ad_email&quot; /&gt;</td>
</tr>
<tr>
<td>Intended Use</td>
<td>&lt;textarea name=&quot;intend&quot; rows=&quot;10&quot;&gt;&lt;/textarea&gt;</td>
</tr>
</tbody>
</table>

<input type="submit" name="Submit" value="Submit" />

<br />
<br />

<table>
<thead>
<tr>
<th>TESTBED</th>
</tr>
</thead>
<tbody>
<tr>
<td>100%</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
var d=new Date();
var monthname=new Array("January","February","March","April","May","June","July","August","September","October","November","December");
//Ensure correct for language. English is "January 1, 2004"
var TODAY = monthname[d.getMonth()] + " " + d.getDate() + ", " + d.getFullYear();
//---------------------------
</script>
</head>
<body bgcolor="#C0DFFD">
<table width="100%" border="0" cellspacing="0" cellpadding="0">
<tr bgcolor="#3366CC" >
<td width="300" colspan="3" rowspan="2"><img src="logo.jpg" alt="Header image" width="374" height="127" border="0" /></td>
<td width="700" height="63" colspan="3" id="logo" valign="bottom" align="left" nowrap="nowrap">SCIBED</td>
<td width="100%"&nbsp;</td>
</tr>
<tr bgcolor="#3366CC">
<td height="64" colspan="3" id="tagline" valign="top" align="left">Web-based Testbed for WSN</td>
<td width="100%"&nbsp;</td>
</tr>
<tr>
<td colspan="7" bgcolor="#003366"><img src="mmspacer.gif" alt="" width="1" height="1" border="0" /></td>
</tr>
<tr bgcolor="#CCFF99">
<td colspan="7" id="dateformat" height="25"> &nbsp;&nbsp;<script language="JavaScript" type="text/javascript">document.write(TODAY); </script> &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</td>
</tr>
<tr>
<td colspan="7" bgcolor="#003366"><img src="mmspacer.gif" alt="" width="1" height="1" border="0" /></td>
</tr>
<tr>
<td width="165" valign="top" bgcolor="#E6F3FF">
<table border="0" cellspacing="0" cellpadding="0" width="165" id="navigation">
<tr>
&lt;td width="165"&gt;&nbsp;&lt;br /&gt;&lt;br /&gt;&lt;/td&gt;&lt;/tr&gt;&lt;/table&gt;&lt;br /&gt;&lt;br /&gt;&lt;br /&gt;&lt;br /&gt;&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td width="50"&gt;&lt;img src="mm_spacer.gif" alt="" width="50" height="1" border="0" /&gt;&lt;/td&gt;&lt;td width="305" colspan="2" valign="top"&gt;&lt;img src="mm_spacer.gif" alt="" width="305" height="1" border="0" /&gt;&lt;br /&gt;&lt;br /&gt;&lt;br /&gt;&lt;table align="center" border="1" cellspacing="0" cellpadding="0" width="305"&gt;&lt;tr&gt;&lt;td colspan="8" class="pageTitle" align="center">Display Jobs &lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td height="36" colspan="2" class="bodyText"&gt;&lt;b&gt;&lt;div align="left">START DATE &lt;/div&gt;&lt;/b&gt;&lt;/td&gt;&lt;td height="36" colspan="2" class="bodyText"&gt;&lt;b&gt;START TIME&lt;/b&gt;&lt;/td&gt;&lt;td height="36" colspan="2" class="bodyText"&gt;&lt;b&gt;FILE&lt;/b&gt;&lt;/td&gt;&lt;td height="36" colspan="2" class="bodyText"&gt;&lt;b&gt;RESULTS&lt;/b&gt;&lt;/td&gt;&lt;/tr&gt;&lt;?php include_once("connection.php");&gt;&gt;$user_name= $_SESSION["user"];
```php
$query = "select * from $user_name";

$result = mysql_query($query);
while($row = mysql_fetch_row($result))
{
    $date=$row[0];
    $time=$row[1];
    $file=$row[2];
    $results=$row[3];
}

<?php

    <tr>
    <td colspan="2" class="bodyText"><?php echo $date;?&gt;</td>
    <td colspan="2" class="bodyText"><?php echo $time;?&gt;</td>
    <td colspan="2" class="bodyText"><?php echo $file;?&gt;</td>
    <td colspan="2" class="bodyText" ><a href="#"><?php echo $results;?&gt;&lt;/a&gt;&lt;/td&gt;&lt;/tr&gt;

    &lt;?php } ?&gt;
    &lt;/table&gt;
    &lt;tr&gt;&lt;/tr&gt;

    &lt;br /&gt;&lt;/td&gt;
    &lt;img src="mm_spacer.gif" alt="" width="50" height="1" border="0" /&gt;&lt;/td&gt;
    &lt;td width="190" valign="top"&gt;&lt;br /&gt;
    &nbsp;&lt;br /&gt;

    &lt;td width="100%"&gt;&nbsp;&lt;/td&gt;
    &lt;/tr&gt;
    &lt;tr&gt;
    &lt;td width="165"&gt;&nbsp;&lt;/td&gt;
    &lt;td width="50"&gt;&nbsp;&lt;/td&gt;
    &lt;td width="167"&gt;&nbsp;&lt;/td&gt;
    &lt;td width="138"&gt;&nbsp;&lt;/td&gt;
    &lt;td width="50"&gt;&nbsp;&lt;/td&gt;
    &lt;td width="190"&gt;&nbsp;&lt;/td&gt;
    &lt;td width="100%"&gt;&nbsp;&lt;/td&gt;
    &lt;/tr&gt;
    &lt;/table&gt;
    &lt;/body&gt;
    &lt;/html&gt;
```
<?php

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Home Page</title>
<script type="text/javascript">
function showUser(str)
{
if (str=="")
{
    document.getElementById("txtHint").innerHTML="";
    return;
}
if (window.XMLHttpRequest)
    {// code for IE7+, Firefox, Chrome, Opera, Safari
        xmlhttp=new XMLHttpRequest();
    }
else
    {// code for IE6, IE5
        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
xmlhttp.onreadystatechange=function()
{
if (xmlhttp.readyState==4 &amp;&amp; xmlhttp.status==200)
    {
        document.getElementById("txtHint").innerHTML=xmlhttp.responseText;
    }
}
xmlhttp.open("GET","getuser.php?q="+str,true);
xmlhttp.send();
}
</script>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<link rel="stylesheet" href="mm_travel2.css" type="text/css" />
<script language="JavaScript" type="text/javascript">

//----Declaring variables----
var d=new Date();

</script>

</head>

<body>

</body>
</html>
var TODAY = monthname[d.getMonth()] + " " + d.getDate() + ", " + d.getFullYear();

\-----------------------------

</script>
</head>
<body bgcolor="#C0DFFF">
<table width="100%" border="0" cellspacing="0" cellpadding="0">
<tr bgcolor="#3366CC">
<td width="382" colspan="3" rowspan="2"><img src="logo.jpg" alt="Header image" width="374" height="127" border="0" /></td>
<td width="378" height="63" colspan="3" id="logo" valign="bottom" align="center" nowrap="nowrap">SCIBED</td>
<td width="100%">&nbsp;</td></tr>
<tr bgcolor="#3366CC">
<td height="64" colspan="3" id="tagline" valign="top" align="center">Web-based Testbed for WSN</td>
<td width="100%">&nbsp;</td>
</tr>
<tr><td colspan="7" bgcolor="#003366"><img src="mm_spacer.gif" alt="" width="1" height="1" border="0" /></td></tr>
<tr bgcolor="#CCFF99">
<td colspan="7" id="dateformat" height="25">&nbsp;&nbsp;<script language="JavaScript" type="text/javascript">
    document.write(TODAY);  
    </script>    </td></tr>
<tr><td colspan="7" bgcolor="#003366"><img src="mm_spacer.gif" alt="" width="1" height="1" border="0" /></td></tr>
<tr><td width="165" valign="top" bgcolor="#E6F3FF">
<table border="0" cellspacing="0" cellpadding="0" width="165" id="navigation">
<tr>
<td width="165"><br />
&amp;nbsp;&lt;br /&gt;&lt;/td>
</tr>
<tr><td width="165"><a href="AdminCreate.php" class="navText">Create

</a></td></tr>
</table></td>
</tr>
</table>

59
<table border="0" cellspacing="0" cellpadding="0" width="305">
<tr>
<td colspan="2" class="pageName">Update User</td></tr>
<tr>
<td width="139" class="bodyText"><p align="left">Username</p></td>
<td width="166" class="bodyText"><label>
<select name="olduser" onchange="showUser(this.value)"
<option value="">Select a User</option>
<?php include_once("connection.php");
$query="select user_name from users";"";}?>
</select>
</label></td></tr>
</table>

echo $query;
$result=mysql_query($query);
while($row=mysql_fetch_row($result)) {
    ?
        <option value="<?php echo $row[0]?>"><?php echo $row[0]?></option>
    <?php }?
</select>
</label></td>

</tr>
<tr>
<td class="bodyText">&nbsp;</td>
<td class="bodyText">&nbsp;</td>
</tr>
<tr>
<td>
    <div id="txtHint"></div>
</td>
</tr>
<tr>
<td class="bodyText">&nbsp;</td>
<td class="bodyText">&nbsp;</td>
</tr><tr>
<td height="32" class="bodyText">&nbsp;</td>
<td class="bodyText"><input type="submit" name="Submit" value="Submit" />
</td>
</tr>
</table>
</form>

<br />
<td width="50"><img src="mm_spacer.gif" alt="" width="50" height="1" border="0" />
</td>
<td width="190" valign="top"><br />
&nbsp;<br />
<table border="0" cellspacing="0" cellpadding="0" width="190">
<tr>
<td colspan="3" class="subHeader" align="center">TESTBED</td>
</tr>
</table>
</td>
runscript.php

```php
<?php
$dir='/home/geetha/local/src/tinyos-2.x/apps/BaseStation/';
chdir($dir);
$data=array();
unset($data);
exec('ls',$data,$ot);
echo "<pre>";
if($ot==0){
    foreach($data as $line){
        echo "$line 
";
    }
} else{
    echo "Error in command";
}
```
}  //echo "</pre>";  //$dir='/home/geetha/local/src/tinyos-2.x/';  //chdir($dir);  //$output=exec('tos-deluge serial@/dev/ttyUSB1:57600 -i 1 tos_image.xml');  //echo "<pre>$output</pre>";  ?>
APPENDIX B. SOFTWARE INSTALLATIONS

The following installations are specific to Ubuntu 8.04 and have been taken from http://mazesolutions.me and http://tinyos.net. This document assumes that Ubuntu is already installed on your machine. Most of the instructions below are valid for Linux versions with minor changes.

I. Installing Apache:

- Open the terminal from Applications -> Accessories -> Terminal
- Type `sudo apt-get install apache2`
- Enter your password when prompted for it

This should download and install Apache2 on your machine. The WWW folder will be in `/var/www/`. To check if the server is running, open a browser and type `http://localhost` in the address bar. This should open up a HTML page. Use the following commands to start, stop or restart Apache Web server:

- Start: `sudo /etc/init.d/apache2 start`
- Stop: `sudo /etc/init.d/apache2 stop`
- Restart: `sudo /etc/init.d/apache2 restart`

II. Installing PHP:

- In the terminal again, type `sudo apt-get install php5 libapache2-mod-php5`
- Restart Apache using `sudo /etc/init.d/apache2 restart`

This should download and install PHP5 on your machine. To test this, create a simple PHP page in `/var/www/` and run it in the browser.
III. Installing MySQL server:

- Again in the terminal, type `sudo apt-get install mysql-server`
- Enter your password when prompted for it

To run MySQL, type the following:

`mysql -uroot -pxxx`

where `xxx` is your password.

IV. Installing PHPmyadmin:

- PHPmyadmin is a graphical user interface of MySQL and can be installed by running the following command in the terminal:

  `sudo apt-get install phpmyadmin`

- The PHPmyadmin configuration file will be in `/etc/phpmyadmin`

- You will need to edit the Apache config file by typing

  `sudo gedit /etc/apache2/apache2.conf` and include the following line:

  `Include /etc/phpmyadmin/apache.conf`

- Restart Apache using `sudo /etc/init.d/apache2 restart`

V. Installing TinyOS

The following instructions show how to download and install TinyOS from CVS repository.

**Step 0:** Enable the Stanford tinyos repository

`sudo echo "deb http://tinyos.stanford.edu/tinyos/dists/ubuntu hardy main" >> /etc/apt/sources.list`
sudo apt-get update

**Step 1:** Install the nesC compiler

`sudo apt-get install nesc`

**Step 2:** Install the crosstools

Debian MSP430: `sudo apt-get install msp430-binutils-tinyos msp430-gcc-tinyos msp430-libc-tinyos`

Debian AVR: `sudo apt-get install avr-binutils-tinyos msp430-gcc-tinyos msp430-libc-tinyos`

**Step 3:** Checkout the TinyOS 2.x source tree

`cd ~`

`mkdir -p local/src`

`cd local/src`

`cvs -z3 -d:pserver:anonymous@tinyos.cvs.sourceforge.net:/cvsroot/tinyos co -P tinyos-2.x`

**Step 4:** Compile the TinyOS tools

`cd tinyos-2.x/tools`

`.Bootstrap`

`.configure --prefix=$HOME/local`

`make all`

`make install`

A few ENTERs will be required by the last two commands.

**Step 5:** Set the environment variables

Add the following to `.profile` (or `.bash_profile` or `.bashrc` depending on your system):
export PATH=$HOME/local/bin:$PATH

export TOSROOT=$HOME/local/src/tinyos-2.x

export TOSDIR=$TOSROOT/tos

export MAKERULES=$TOSROOT/support/make/Makerules

export CLASSPATH=$TOSROOT/support/sdk/java/tinyos.jar:

export PYTHONPATH=.:$TOSROOT/support/sdk/python:$PYTHONPATH

export PATH=$HOME/local/src/tinyos-2.x/support/sdk/c:$PATH
APPENDIX C. TESTBED SETUP INSTRUCTIONS

The following instructions show how to quickly setup an experimental testbed for Wireless Sensor Networks. Please refer to Appendix B for instructions to install the required software.

Testbed Specifications:

- Host OS: Ubuntu 8
- Hardware: Micaz motes, MIB 520, USB cable
- Software: TinyOS-2.x, Apache server, PHP, MySQL
- Languages: nesC

Step 1: Develop Web Interfaces

We have developed a set of Web Interfaces for both Users and Administrators. We have separate login pages for users and administrators to avoid security breaches. Appendix A provides the source code for all Web pages. These include PHP scripts interfacing with databases. In order to test the Web pages, please follow the instructions in Appendix B to install Apache, PHP and MySql and copy the PHP files into the WWW folder on the Apache server. To run the files, open a browser and type http://localhost/page where page is the actual PHP file name.

Step 2: Create Databases

We created a Scibed database with one table named “users”. This table has two fields “user_name” and “password”. This table is used to store the login credentials of
authorized users. Another table named after the user_name is automatically created when the administrator makes an entry for that user in the “users” table.

We also created another database called “admin”. This database has one table called “adminlogin” which holds the login credentials of authorized administrator(s).

**Step 3: Develop Perl Script**

A perl script is needed to automate the dissemination and collection functions described below.

**Automation of program uploading:**

To automate the dissemination function, include some functionality on the “run” button of “getuserinfo.php” page. Clicking on the button should invoke a script that takes the uploaded .xml file as input and runs the following commands on it to automatically inject and disseminate the image to the Base station and Client motes.

```plaintext
chdir(~/.local/src/tinyos-2.x/)
tos-deluge serial@/dev/ttyUSB1:57600 –i 1 <image file>
tos-deluge serial@/dev/ttyUSB1:57600 –d 1 <image file>
tos-deluge serial@/dev/ttyUSB1:57600 –dr 1 <image file>
```

For each command run above, there should be an output file named significantly and uniquely.

**Automation of data querying:**

A similar script is required to automate data collection. A “Results” button has to be included on the “manage.php” webpage. When the button is clicked, it has to invoke a
script that pulls the data stored as a file and display it to the user as a file. The script should include the following commands:

```bash
chdir(~/.local/src/tinyos-2.x/)
java.net.tinyos.tools.Listen
```

**Step 4: Run the Dissemination Component**

The dissemination component is a six steps process as described below. The dissemination is achieved via a protocol called Deluge T2. Also in this section, we discuss the command utility of the Deluge T2 protocol.

1. **Boot process**: The first step is compile a “tosboot” application which is a bootloader for deluge T2. To compile the tosboot, navigate to `/tinyos-2.x/tos/lib/tosboot` and run the command `make micaz`

2. **Preparing the Base Station**: The second step is to compile and install “BaseStation” application on one of the motes, which will then act as the base station for the rest of this exercise. To compile and install this application, connect a mote to the programming board and the board has to be connected to the computer via a USB bus. Then, navigate to `/tinyos-2.x/apps/tests/deluge/BaseStation` and run the compile and install command: `make micaz install,0 mib520,/dev/ttyUSB0` . The BaseStation application has no function other than initializing the deluge component on the base station.

3. **Preparing the Clients**: The third step is to initialize the deluge component on the client motes. To do so, we install the “GoldenImage” application on all the motes in the network. Connect the mote to the programming board and while in the
terminal, navigate to /tinyos-2.x/apps/tests/deluge/GoldenImage . Type in the command: make micaz install,1 mib520,/dev/ttyUSB0 . Make sure to change the identity every time you install the GoldenImage on a new node.

4. **Preparing the Application:** The fourth step involves preparing the application that needs to be disseminated. There are three things to be verified in the application:

   - The main application file should include the Deluge component i.e., it must include the following statement: `Components DelugeC`
   - The make file should include a statement to initialize the bootloader:
     ```
     BOOTLOADER = tosboot
     ```
   - There should be a “volumes-at45db.xml” file in the application folder. This file defines the storage volumes by name and size in bytes.

5. **Inject the Program Image:** Once the application is ready, the next step requires injecting the application into the base station. The command used for this is tos-deluge. To download the image file to the base station, run the following command:

   ```
   tos-deluge serial@/dev/ttyUSB1:57600 –i 1 apps/…/build/micaz/tos_image.xml
   ```

   where “…” is the application folder. This command injects the program image file into the base station on the flash volume labeled 1.

6. **Issuing the Disseminate Command:** The final step is to disseminate the file on the base station to the remaining nodes. This is using the tos-deluge command again:

   ```
   tos-deluge serial@/dev/ttyUSB1:57600 –d 1
   ```
This command informs the base station to advise all the nodes in the network about the availability of a new program image.

**Deluge Command Utility:**

Deluge offers a simple command line utility [TOS 2010] that allows a user to perform multiple queries.

- **-p**: this option provides information about the images currently executing and installed on a particular mote. The information given includes program name, user ID, Compiled date and time, hardware platform, host name, user hash, size and number of pages.
- **-i**: this option takes the tos_image.xml file as input and creates and injects an image file into the specified flash volume of the mote.
- **-d**: this option informs the base station to notify the notes in the network about the availability of a new program image.
- **-dr**: this option is for both dissemination and running the image file and forces a reboot.
- **-e**: this option is used to erase a volume on the flash memory of the base station.
- **-s**: this option stops the propagation of image file any further.
- **-ls**: this option is used to unlock a volume to perform any operations on it.

**Step 5: Run the Collection Component**

Data collection is accomplished via a Listener tool offered by TinyOS. It is a simple three step process as described below:
1. **Test Communication:** The first step in setting up the serial forwarder is to check the communication between the PC and the mote by installing ‘TestSerial’ application. This communication is achieved through serial port.

2. **Set the Serial Port:** In the next step, run the associated java application that communicates with the serial port. To do so, run the following command: `java TestSerial –comm serial@/dev/ttyUSB1:57600`. This command instructs the application to listen to the serial port.

3. **Use the Listen Tool:** Finally, we use the listen tool to print out the packets that it sees. The command to use Listen is: `Java net.tinyos.tools.Listen`

The output of the Listen tool has to be saved to a file named uniquely. Each file uploaded by a user has a corresponding output file that is stored for the user to be retrieved later.