

University of New Orleans
ScholarWorks@UNO

University of New Orleans Theses and
Dissertations

Dissertations and Theses

5-16-2008

Implementation of Secure Key Management Techniques in Wireless Sensor Networks

Noor Ottallah
University of New Orleans

Follow this and additional works at: <https://scholarworks.uno.edu/td>

Recommended Citation

Ottallah, Noor, "Implementation of Secure Key Management Techniques in Wireless Sensor Networks" (2008). *University of New Orleans Theses and Dissertations*. 703.
<https://scholarworks.uno.edu/td/703>

This Thesis is protected by copyright and/or related rights. It has been brought to you by ScholarWorks@UNO with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you need to obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself.

This Thesis has been accepted for inclusion in University of New Orleans Theses and Dissertations by an authorized administrator of ScholarWorks@UNO. For more information, please contact scholarworks@uno.edu.

Implementation of Secure Key Management Techniques in Wireless Sensor Networks

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Masters of Science
In
Computer Science
Information Assurance

By

Noor J. Ottallah

B.S University of New Orleans, 2005

May, 2008

Acknowledgment

Any project no matter how small or big is never worked on solo. For all the people who stood by me to make this project happen, I take the time to thank them. I feel greatly privileged to express thanks to all the people who helped me to complete the project successfully. I would like to thank Dr. Jing Deng, Professor, mentor, and friend of the Computer Science Department, University of New Orleans, for giving me the opportunity and infrastructure to work on this project and for providing his expertise and guidance throughout the duration of the project. Big thanks to Fareed Qadura director of the I.T operations for extending his support and expertise to me. I convey my sincere regards to our internal guide Dr Shengru Tu, for providing invaluable suggestions and guidance at all stages of the project. I also would like to thank Dr. Adlai Depano for being on my research committee. A Big thank you to all the faculty members of the CSCI department for teaching me what I know. I would also like to thank the Sermis development team, at the Technology Park for their respect and support and understanding. With out it I might have not had time to finish this project.

Last but definitely not least, to my wife Shirouk, your love and your constant encouragement kept me going when I felt like quitting. I love you and I know you will always be there for me.

Contents

Abstract.....	v
Chapter 1 Introduction	1
1.1 Communications in WSNs.....	2
1.2 Security issues in WSNs.....	4
1.3 Security Requirements for WSNs.....	5
1.4 Security Key Issues	7
1.5 Key Distribution.....	8
1.5.1 Key pre-distribution	8
Chapter 2 System Requirement Specification	10
2.1 Hardware Specification	11
2.1.1 MPR2400 MICAz Mote.....	11
2.1.2 The MTS300 Sensor Boards	16
2.2 Software Specifications.....	19
2.2.1 TinyOS	20
2.2.2 NesC	21
2.2.3 TinyDB	23
2.3 Software Installation.....	27
2.3.1 Pc Requirements.....	27
2.3.2 Installation of apps on the Nodes.....	36
2.3.3 Connecting the hardware	37
2.4 Installing TinyDB application.....	40
2.4.1 Starting the TinyDB GUI on the PC.....	41
2.4.2 Displaying the Topology.....	41
2.4.3 Simple Queries	43

2.4.4 Results being received and charted by TinyDB.....	44
2.5 Storing the Results Data in PostgreSQL (Optional)	44
Chapter 3 System Implementation	47
3.1 Design.....	49
3.2 Implementation	51
3.2.1 TinyOS changes	51
3.2.2 Java Application changes	54
3.3 Securing TinyDB+	56
3.4 Changing the Keys in TinyDB+.....	57
3.4.1 Key updating using TinySec.....	57
3.4.2 Key updating using Over the Air reprogramming.....	58
Chapter 4 Tests and Results.....	60
4.1 Connectivity	60
4.2 Security.....	66
Chapter 5 Conclusion and future work	67
References	68
Vita	71

Abstract

Creating a secure wireless sensor network involves authenticating and encrypting messages that are sent throughout the network. The communicating nodes must agree on secret keys in order to be able to encrypt packets. Sensor networks do not have many resources and so, achieving such key agreements is a difficult matter. Many key agreement schemes like Diffie-Hellman and public-key based schemes are not suitable for wireless sensor networks. Pre-distribution of secret keys for all pairs of nodes is not viable due to the large amount of memory used when the network size is large.

We propose a novel key management system that works with the random key pre-distribution scheme where deployment knowledge is unknown. We show that our system saves users from spending substantial resources when deploying networks. We also test the new system's memory usage, and security issues. The system and its performance evaluation are presented in this thesis.

Key Words

Wireless sensor networks, TinyDB, node, secure key, TinyOS, NesC.

Chapter 1 Introduction

Wireless sensor networks are networks of small, battery-powered, memory-constraint devices named sensor nodes, which have the capability of wireless communication over a restricted area [1]. Due to memory and power constraints, they need to be well arranged to build a fully functional network. Environments, where sensor nodes are deployed, can be controlled (such as home, office, warehouse, forest, etc.) or uncontrolled (such as hostile or disaster areas, toxic regions, etc.). When the environment is known and under control, deployment may be achieved manually to establish an infrastructure. However, manual deployments become infeasible or even impossible as the number of the nodes increases. If the environment is uncontrolled or the WSN is very large, deployment has to be performed by randomly scattering the sensor nodes to target area. It may be possible to provide denser sensor deployment at certain spots, but exact positions of the sensor nodes can not be controlled [2]. Thus, network topology can not be known precisely prior to deployment. Although topology information can be obtained by using mobile sensor nodes and self-deployment protocols as proposed in [3], this may not be possible for a large scale WSN.

To insure the security factor in WSNs we must tackle five challenges [3, 4]:

1. Wireless nature of communication.
2. Resource limitation on sensor nodes.
3. Very large and dense WSNs and lack of a central infrastructure.
4. Unknown network topology prior to deployment.
5. High risk of physical attacks to unattended sensors.

Many solutions for Network Security for networks that operate under adversarial conditions depend on the existence of strong and efficient key distribution techniques. However, it is very difficult, or even impossible in uncontrolled environments, to visit large number of sensor nodes, and change their setup. Moreover, use of a unary shared key that would be hard coded into all of the nodes in a WSN is not a good idea because any hostile force can easily obtain the key. To overcome this problem, sensor nodes have to adapt their environments, and create or find a secure network by:

1. Use pre-distributed keys or key algorithms,
2. Exchange information with their closest neighbors
3. Exchange information with computationally strong nodes [5].

1.1 Communications in WSNs

Communication in WSNs is much like wireless ad hoc networks. Similarly, WSNs are ever changing in a way that radio range and network connectivity changes by time. Sensor nodes die and new sensor nodes may come up in to the network. However, WSNs are more constrained, denser, and have to deal with redundant information which can be good or bad depending on the application. WSN architectures are organized in hierarchical and distributed structures as shown in Figure 1.

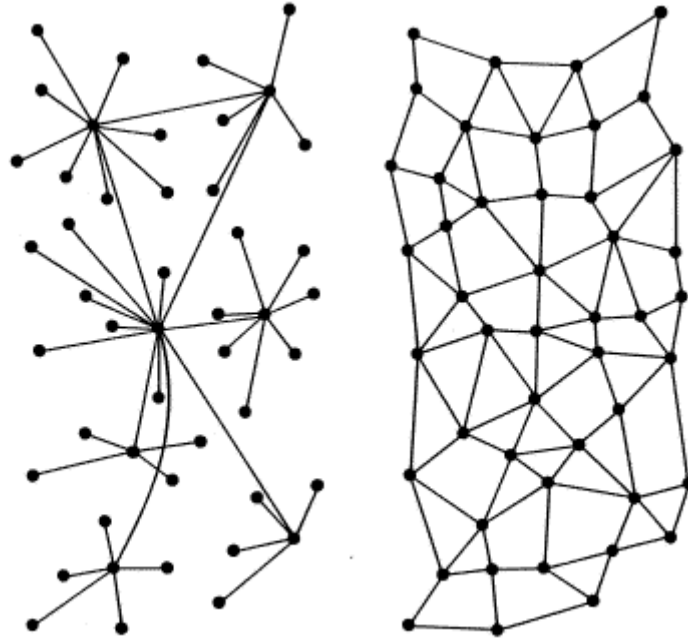


Fig. 1 Network Models: Hierarchical and Distributed Wireless Sensor Networks.

Hierarchical WSNs as shown in Figure 1(left); there is a hierarchy among the nodes based on their capabilities: base stations or root nodes, cluster heads and sensor nodes. Base stations are on a higher magnitude of power than sensor nodes and cluster heads. A base station is typically a gateway to another network, a powerful data processing / storage center, or an access point for human interface. In some cases the base station is given the responsibility of key distribution.

A Distributed WSNs is shown in Figure 1(right); there is no preset infrastructure, and network topology is not known prior to deployment. Sensor nodes are usually randomly scattered all over the target area. Once they are deployed, each sensor node scans its radio coverage area to figure out its neighbors. Data flow in distributed WSNs is similar to data flow in Hierarchical WSNs with a difference that network (broadcasts) can be sent by all the sensor nodes.

1.2 Security issues in WSNs

Hostile forces have better chances at attacking WSNs because of the lack of infrastructure and lack of a controlled environment. Equipped hackers with powerful computers and communication devices may access the whole WSN from a remote location which can be stationed any where around the network. They can gain mobility by using powerful laptops, batteries and antennas, and move around or within the WSN. Also, adversaries can plant their own sensor nodes, base stations or cluster heads in uncontrolled environments. They can compromise, replace or physically damage existing ones. Wirelessly attacking WSNs can help adversaries to perform different variety of active, passive and stealth type of attacks [5].

●Passive eavesdropping

An unauthorized node will be able to gather data that could include the information of the network topology, and other information about other authorized nodes which forward or receive data. Hence, techniques may be needed to hide such information. Eavesdropping is also a threat to location privacy. Note that passive eavesdropping also allows unauthorized nodes to discover the exact geographical location of a node by just detecting and calculating the signal strength.

●Active interference

The major threat from active interference is a denial of service attack caused by blocking the wireless communication channel. The effects of such attacks depend on the routing protocol in use, for example, reactive routing protocols may detect a denial of service attack as a link break. Route maintenance operations, such as, “RTS messages” will cause most protocols to report the link as broken to the other nodes in the environment.

The content of data that flows with in WSNs can be defined as:

1. Mobile code.
2. Sensor Key readings.
3. Key management.
4. Location information.

In addition to active and passive attacks on key management traffic, adversaries may improve their capabilities by accessing mobile codes and location information. An adversary can insert a malicious mobile code which might spread to whole WSN, potentially compromising its security. It can use the location information to locate critical nodes, capture and read their security contents [4, 7].

1.3 Security Requirements for WSNs

To provide security, encryption technologies are used to achieve secret communications. To encrypt the data, a prerequisite is the secret keys should be set up among communicating sensor nodes. Key management is the process in which keys are created, stored, protected, transferred, used and destroyed [8]. Keying means the process of achieving keys agreement among sensor nodes by deriving common secret keys among communicating parties. Pair-wise keying involves two parties agreeing on a shared session key while group keying is that more than two parties to set up the communication key [9]. Currently several keying schemes have been propose [10, 11, 12, and 13]. The following table sums up the three different protocols:

Scheme	Algorithm	Trade-offs
The arbitrated method	Relies on some trusted central point like a Server.	Vulnerable to single point failure, also the ad hoc attribute of WSN makes it difficult to set up a network-wise available
The self-enforcing scheme	Uses the asymmetric encryption cryptography.	Limited by the current computation abilities and energy resources of sensor network technologies
The Pre-distribution key management scheme	Keys are loaded into sensor nodes before deployment	Different methods of loading have different tradeoffs

Table 1 Key distribution schemes

In our research we mainly consider applying the pre distribution key management scheme, in which keys are loaded into sensor nodes before deployment. WSNs which resemble adhoc and MANET networks also need to follow general security requirements which are referred to as the CIAAN [5]:

- Availability: ensuring that service offered by whole WSN, by any part of it, or by a single sensor node must be available whenever required.
- Authentication: authenticating other nodes, cluster heads, and base stations before granting a limited resource, or revealing information.
- Integrity: ensuring that message or the entity under consideration is not altered.
- Confidentiality: providing privacy of the wireless communication channels to prevent eavesdropping.
- Non-reputation: preventing malicious nodes to hide their activities.

1.4 Security Key Issues

Key management is a fundamental security issue in sensor networks. It is the basis to establish the secure communication using cryptographic technologies between sensor nodes in a sensed area. However, because of the limited resources a sensor has, it is infeasible to use traditional key management techniques such as public key cryptography or key distribution center based protocols [14].

A proper key-management service is required to ensure that nodes which are legitimate members of the network will only be those who are equipped with the necessary keys. The following shows several needed classifications of key management properties [4, 6, 9, and 15].

- Efficiency: storage, processing and communication limitations must be considered.
- Storage complexity: amount of memory required to store security credentials.
- Processing complexity: amount of processor cycles required to establish a key.
- Scalability: ability to support larger networks, and must be flexible against substantial increase in the size of the network even after deployment.
- Communication complexity: number of messages exchanged during a key generation process.
- Resilience: resistance against node capture.
- Key connectivity (probability of key-share): probability that two (or more) sensor nodes store the same key or keying material. Enough key connectivity must be provided for a WSN to perform its intended functionality.

1.5 Key Distribution

Key distribution is an important issue in wireless sensor network (WSN) design. There are many key distribution schemes out there that are designed to maintain an easy and at the same time secure communication among sensor nodes. The most accepted method of key distribution in WSNs is key pre-distribution, where secret keys are placed in sensor nodes before deployment. When the nodes are deployed over the target area, the secret keys are used to create the network.

1.5.1 Key pre-distribution

The current key pre-distribution schemes used these days can be classified into four classes which are detailed with mathematical equations in:

- Pure probabilistic key pre-distribution schemes.
- Polynomial-based key pre-distribution schemes [16].
- Blom's matrix-based key pre-distribution schemes.
- Deterministic key pre-distribution schemes.

Basically a key pre-distribution scheme has three phases, key distribution, shared key discovery and path-key establishment [16, 17, and 18]. During these phases, secret keys are generated, placed in sensor nodes, and each sensor node searches the area in its communication range to find another node to communicate.

A secure link is established when two nodes discover one or more common keys, and communication is done on that link between those two nodes [4]. Afterwards, paths are established connecting these links, to create a connected graph. The result is a wireless communication network functioning in its own way.

Each type of scheme has its advantages and application environment. An obvious rule can be drawn from these proposed schemes is that the location knowledge can be used to improve the performance of the key management schemes, such as the connectivity, resilience against nodes capture and memory efficiency [12,19]. Our system design will not care if location knowledge is known or not. It is good to note that most of the key management solutions for wireless sensor networks are trying to find the better tradeoffs between system security (e.g., resilience to node capture) and network connectivity. All of them have weak and strong points.

Chapter 2 System Requirement Specification

In order to understand and test how our system works it is crucial to talk about the components we used to implement the system. Mote kits are a complete set of hardware and software that include programmable sensor nodes, sensor boards, programming boards, cables, batteries, cases, and all the software needed to program the nodes.

To use the mote kit and start programming the nodes, certain programs need to be installed and specific procedures need to be followed. This can be very difficult to do as there is no unified instruction set that tells the user how to do it. This results in wasted time, frustration and quitting. The learning curve is very hard and it is almost best that a user has previous knowledge of UNIX commands, java compiling as well as some knowledge of C language.

This chapter will act as a full detailed instruction tutorial on how to set up the hardware and set up the software needed for a researcher to start programming for wireless nodes. It will also attempt to try to give a full description of all the components and software used along with plenty of figures and visual aids. The information here has been compiled from many sources as well as from our experiments in the lab. We will be using the MICAZ-MOTE 2400CA kit from Cross-Bow Technologies [20].

2.1 Hardware Specification

When you open the box you will need to identify all the components. The kits come with the following:

- 7 MICAz wireless nodes each with 2 AA batteries.
- 3 MTS300 Sensor Boards (Light, Temperature, Acoustic, and Sounder).
- 1 MIB510 Programming and Serial Interface Board.
- Mote-Test Software CD.
- Tiny OS Getting Started Guide PDF.
- 1 MIB600 programming network Interface board with power adaptor.
- 1 interface board to connect other electronic devices to the nodes.

2.1.1 MPR2400 MICAz Mote

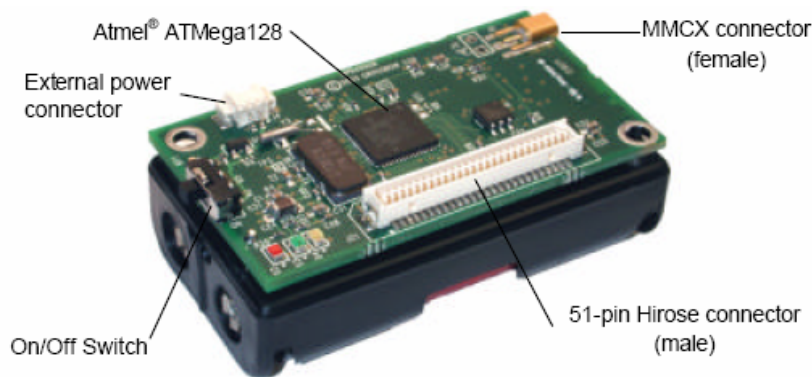


Figure 2 MICAz MPR2400

The MICAz is a 2.4GHz, IEEE 802.15.4 compliant, Mote module used for enabling low-power, wireless, and sensor networks. The MICAz Mote features several new capabilities that enhance the overall functionality of Crossbow's MICA family of wireless sensor networking products. These features include:

1. IEEE 802.15.4/ZigBee compliant RF transceiver.
2. 2.4 to 2.4835 GHz, a globally compatible ISM band.
3. Direct sequence spread spectrum radio which is resistant to RF interference and provides inherent data security.
4. 250 kbps data rate.
5. Runs Tiny OS 1.1.7 and higher, including Crossbow's reliable mesh networking stack software modules.
6. Plug and play with all of Crossbow's sensor boards, data acquisition boards, gateways, and software.
7. It has an Atmel processor, ATmega128L.
8. It has an Atmel Nonvolatile memory of 512KB flash, capable to store up to 100,000 measurements.
9. The radio Transceiver used is the Chipcon CC2420. The board also offer enhanced processor capabilities, including a boot-loader that allows for over air reprogramming.

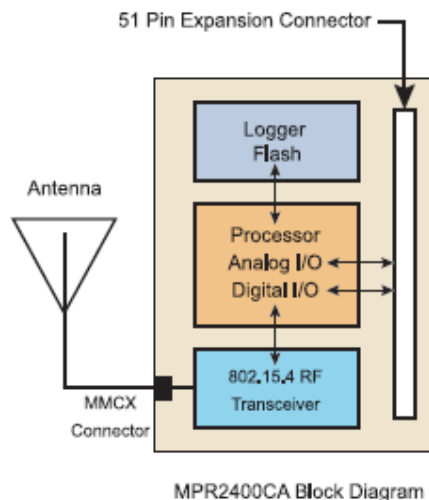


Figure 3 MICAz block diagram

The following table shows some extra features of the MPR2400 also all the information here can also be obtained from Cross-Bow technologies:

Processor/Radio Board	MPR2400CA	Remarks
Processor Performance		
Program Flash Memory	128K bytes	
Measurement Serial Flash	512K bytes	>100,000 measurements
Configuration EEPROM	4 K bytes	
Serial Communications	UART	0 V to 3 V transmission levels
Analog to Digital Converter	10 bit ADC	8 channels, 0 V to 3 V input
Other Interfaces	Digital I/O,I2C,SPI	
Current Draw	8 mA	Active mode
	< 15uA	Sleep mode
RF Transceiver		
Frequency band ¹	2400 MHz to 2483.5 MHz	ISM band, programmable in 1 MHz steps
Transmit (TX) data rate	250 kbps	
RF power	-24 dBm to 0 dBm	
Receive Sensitivity	-90 dBm (min), -94 dBm (typ)	
Adjacent channel rejection	47 dB	+ 5 MHz channel spacing
	38 dB	- 5 MHz channel spacing
Outdoor Range	75 m to 100 m	1/2 wave dipole antenna, LOS
Indoor Range	20 m to 30 m	1/2 wave dipole antenna
Current Draw	19.7 mA	Receive mode
	11 mA	TX, -10 dBm
	14 uA	TX, -5 dBm
	17.4 mA	TX, 0 dBm
	20 uA	Idle mode, voltage regular on
	1 uA	Sleep mode, voltage regulator off
Electromechanical		
Battery	2X AA batteries	Attached pack
External Power	2.7 V - 3.3 V	Molex connector provided
User Interface	3 LEDs	Red, green, and yellow
Size (in)	2.25 x 1.25 x 0.25	Excluding battery pack
(mm)	58 x 32 x 7	Excluding battery pack
Weight (oz)	0.7	Excluding batteries
(grams)	18	Excluding batteries
Expansion Connector	51 pin	All major I/O signals

Figure 4 – Features of MICAZ

2.1.1.1 Powering the Mote

The MPR2400 is powered by two AA batteries; however any battery combination (AAA, C, D, etc) can be used as long as that the output is between 2.7-3.3VDC. Battery life will vary depending on the mote utilization and the battery size used .The battery life usually varies between 1.45 and 17.35 months. The MICAz can also be externally be powered using any wall adaptor (usually comes with the kit).

2.1.1.2 Radio/Antenna Considerations

The MICAZ's CC2420 radio can be tuned within the IEEE 802.15.4 channels that are numbered from 11 (2.405 GHz) to 26 (2.480 GHz) each separated by 5 MHz. The channel can be selected at run-time with the TOS CC2420Radio library call CC2420Control Tune Preset (uint8_t chnl) which is part of the TinyOS folder. By default channel 11 (2480 MHz) is selected.

RF transmission power is programmable from 0 dBm (1 mW) to -25dBm. Lower transmission power can be advantageous by reducing interference and dropping radio power consumption from 17.5 mA at full power to 8.5 mA at lowest power. RF transmission power is controlled using the TOS *CC2420Radio* library call CC2420Control. The Tiny OS component *VoltageM.nc* can be wired into an application to provide this measurement capability. The reserved keyword TOS_ADC_VOLTAGE_PORT is mapped to ADC Channel 30 in the MICAZ.

2.1.1.3 Data Logger

The MICAZ mote also features a 4-Mbit serial flash for storing data, measurements, and other user-defined information. Tiny OS supports a micro file system that runs on top of this flash/data logger component. The serial flash device supports over 100,000 measurements readings. The MICAZ also has a 64-bit serial ID chip.

2.1.1.4 Battery Voltage Monitor

Since the eight-channel, Atmega128 ADC uses the battery voltage as a full scale reference, the ADC full scale voltage value changes as the battery voltage changes. In order to calibrate the battery voltage, a precision external voltage reference is required. The MICAZ uses an LM4041, 1.223 volt reference (V_{ref}) attached to ADC channel 7.

2.1.1.5 The MIB510 Programming and Serial Interface Board

The MIB510 interface board is a multipurpose interface board used with MICAz and MICA2DOT family of products since it has connectors for both MICAz and MICA2DOT. It supplies power the device through an external power adapter option, and provides an interface for a RS-232 mote serial port and reprogramming port.

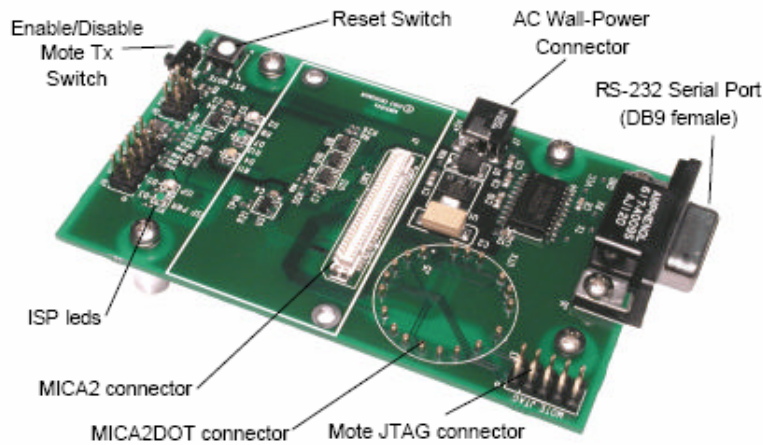


Figure 5 MIB510 programmer

2.1.1.6 In System Processor

The MIB510 has an on-board in-system processor (ISP) to program the motes. The Code is downloaded to the ISP, over the serial port, and the ISP programs the code into the mote. The ISP and mote share the same serial port. The ISP runs at a fixed baud rate of 115Kbaud. The ISP continually monitors incoming serial packets for a special multi-byte pattern. Once this pattern is detected, it disables the mote's serial Rx and Tx, and then takes control of the serial port. The ISP processor has two LEDs, 'SP PWR' (green) and ISP (red). SP PWR is used to indicate the power state of the MIB510 (see below). If the ISP led is on, the MIB510 has control of the serial port. It will blink when the RESET button is activated.

2.1.1.7 Power

The MIB510 has an on-board regulator that will accept 5 to 7 VDC, and supply a regulated 3V to the MICAZ. The MIB510 is delivered with a wall power supply. There is a built-in low voltage monitor that disables reprogramming if the power supply voltage is dangerously low. When the proper programming voltage exists the ISP PWR led is on. If the voltage goes below 2.9 V, the ISP PWR led will blink and disable the mote from any code downloads. If the voltage is too low to power the ISP then the ISP PWR led will be off. Things to remember:

- When programming a MICAZ with the MIB510, turn off the battery switch. The MICAZ does not have switching diodes to switch between external and battery power.
- The RST MOTE switch resets both the ISP and mote processors. RST resets the ISP; after the ISP powers-up it resets the mote's processor.

2.1.2 The MTS300 Sensor Boards



Figure 6 MTS300 sensor board

The MTS300CA are sensor boards with a variety of sensing capabilities. These varieties can be used in developing sensor networks for a variety of applications including movement and acoustic (sound), low-performance seismic sensing, vehicle detection, ranging, robotics, and other applications. The MTS300 Sensor Board features:

- Light
- Temperature
- Microphone, Sounder
- Tone Detection Circuit

2.1.2.1 Light

The light sensor, CL94L, is a simple CdSe photocell, manufactured by Clairex. The maximum sensitivity of the photocell is at the light wavelength of 690nm. In order to use the light sensor, the digital control signal, PW1, must be turned on and the output of the sensor is connected to the analog-digital converter channel 1 (ADC1). When the sensor is exposed to light, the typical on resistance is 2Kohm and the nominal circuit output is near VCC or full-scale.

2.1.2.2 Temperature

The temperature sensor is a surface mount thermostat made by Panasonic It is configured in a simple voltage divider circuit with a nominal mid-scale reading at 25°C. The output of the temperature sensor circuit is available at ADC1. Since the light and temperature sensor share the same A/D converter channel (ADC1). Only turn one sensor on at a time, or the reading at ADC1 will be corrupted and meaningless.

2.1.2.3 Microphone

The microphone circuit has two principal uses. The first use is for acoustic ranging. The second use is for general acoustic recording and measurement.

The basic circuit consists of a pre-amplifier (U1A-1), second-stage amplified with a digital-pot control (U1A, PT2). Audio files have been recorded into the Logger Flash memory of MICAZ Motes for later download and entertainment (or analysis). The second stage output (mic_out) is routed through an active filter (U2) and then into a tone detector (TD1). The LM567 CMOS Tone Detector IC actually turns the analog microphone signal into a digital high or low level output at INT3 when a 4 KHz tone is present .

The Sounder circuit on the sensor board can generate this tone. A novel application of the sounder and tone detector is acoustic ranging. In this application, a mote pulses the sounder and sends an RF packet via radio at the same time. A second mote listens for the RF packet and notes the time of arrival by resetting a timer/counter on its processor. It then increments a counter until the tone detector detects the sounder. The counter value is the Time-of-Flight of the sound wave between the two motes.

The Time-of-Flight value can be converted into an approximate distance between motes. Using groups of Motes with Sounders and Microphones, a crude localization and positioning system can be built using this technique.

2.1.2.4 Sounder

The sounder is a simple 4 kHz fixed frequency piezoelectric resonator. The drive and frequency control circuitry is built into the sounder. The only signal required to turn the sounder on and off, is Sounder_Power. Sounder_Power is controlled through the power control switch (P1) and is set by the hardware line PW2.

2.1.2.5 Turning Sensors On and Off

All of the sensors have a power control circuit. The standard condition for the sensor is to be off. This design helps minimize power consumption by the sensor board. In order to turn sensors on; control signals are issued to the power switches.

2.2 Software Specifications

TinyOS is a free and open source component-based operating system and platform targeting wireless sensor networks (WSNs). TinyOS is an embedded operating system written in the nesC programming language as a set of cooperating tasks and processes [21]. TinyOS started as collaboration between the University of California and in co-operation with Intel Research. It has since grown to be an international consortium, the TinyOS Alliance. We build our system around the following software components:

- TinyOS supports Crossbow Mica motes, MICAz motes and Mica2Dot motes and a few other wireless sensor devices. TinyOS is embedded in Crossbow motes and therefore we will use TinyOS structures for developing the system.

- NesC (network embedded systems C) is the Programming language used to develop software on TinyOS. NesC is an extension to C that involves the necessary structures and concepts to support event-driven execution of TinyOS.
- TinyDB: TinyDB [21, 22] which is a query processing system for extracting information from a network of TinyOS sensors. In addition to the mote software, TinyDB provides a PC interface written in Java. The TinyDB software is to be modified to incorporate the Key management algorithm.

2.2.1 TinyOS

TinyOS is an open-source operating system designed for wireless embedded sensor networks. It is made up from a component-based architecture which enables rapid innovation and implementation while minimizing code size as required by the severe memory limitation inherent in sensor networks [21]. TinyOS's component library includes network protocols, distributed services, sensor drivers, and data acquisition tools - which can be used as-is or be further refined for a custom application. TinyOS's event-driven execution model enables fine-grained power management yet allows the scheduling flexibility made necessary by the unpredictable nature of wireless communication and physical world interfaces. TinyOS has been ported to over a dozen platforms and numerous sensor boards. A wide community uses it in simulation to develop and test various algorithms and protocols [21, 23].

Everything in a TinyOS application is static:

- No Dynamic Memory (no malloc ())
- No Function Pointers
- No Heap

This means that just about everything is known at compile time by the NesC compiler. This allows the compiler to perform global compile time analysis to detect data race conditions, and where function in lining will improve performance. This relieves the developer of these burdens and hence development is made easier and the systems robustness is improved. The memory map of a TinyOS application is similar to the structure of an executable image on the Unix OS. The MICAz generation of Motes which we use in our project consists of 128k of program flash and 4k of SRAM. The memory image is as follows:

In the 128K Program Flash

- "text" section holds the executable Code
- "data" section holds the program Constants

In the 4K SRAM (flash)

- "bss" section holds the variables
- The rest of the bss is free space – which means it is fixed because the lack of dynamic memory.
- stack - grows down in the free space

2.2.2 NesC

NesC is an extension to the C programming language designed to embody the structuring concepts and execution model of TinyOS .TinyOS is an event-driven operating system designed for sensor network nodes that have very limited resources (e.g., 8K bytes of program memory, 512 bytes of RAM).

The basic concepts behind NesC are:

- Separation of construction and composition: programs are built out of components, which are assembled (“wired”) to form whole programs. Components have internal concurrency in the form of tasks. Threads of control may pass into a component through its interfaces. These threads are rooted either in a task or a hardware interrupt.
- Specification of component behavior in terms of set of interfaces: Interfaces may be provided or used by components. The provided interfaces are intended to represent the functionality that the component provides to its user; the used interfaces represent the functionality the component needs to perform its job.
- Interfaces are bidirectional: they specify a set of functions to be implemented by the interface’s provider (commands) and a set to be implemented by the interface’s user (events). This allows a single interface to represent a complex interaction between components (e.g., registration of interest in some event, followed by a callback when that event happens). This is critical because all lengthy commands in TinyOS (e.g. send packet) are non-blocking; their completion is signaled through an event (send done). By specifying interfaces, a component cannot call the send command unless it provides an implementation of the send-Done event.
- Typically commands call downwards, i.e., from application components to those closer to the hardware, while events call upwards. Certain primitive events are bound to hardware interrupts.
- Components are statically linked to each other via their interfaces. This increases runtime efficiency, encourages robust design, and allows for better static analysis of programs.
- NesC is designed under the expectation that code will be generated by whole program compilers. This should also allow for better code generation and analysis.

2.2.3 TinyDB

TinyDB [22, 24] is a query processing system for extracting information from a network of TinyOS sensors. TinyDB does not require writing embedded C code for sensors like TinyOS. Instead, TinyDB provides a simple, SQL-like interface to specify the data, along with additional parameters. It functions like posing queries against a traditional database. Given a query specifying data interests, TinyDB collects that data from motes in the environment, filters it, aggregates it together, and routes it out to a PC or a Base Station. TinyDB does this via power-efficient in-network processing algorithms. To use TinyDB, its TinyOS components are installed onto each mote in the sensor network. TinyDB provides a simple Java API for writing PC applications that query and extract data from the network; it also comes with a simple graphical query-builder and result display that uses the API. The primary goal of TinyDB is to make work as a programmer significantly easier, and allow data-driven applications to be developed and deployed much more quickly than what is currently possible. TinyDB frees from the burden of writing low-level code for sensor devices, including the (very tricky) sensor network interfaces. Some of the features of TinyDB include:

- **Metadata Management:** TinyDB provides a metadata catalog to describe the kinds of sensor readings that are available in the sensor network.
- **High Level Queries:** TinyDB uses a declarative query language that describes the data wanted, without requiring saying how to get it. This makes it easier to write applications, and helps guarantee that the applications continue to run efficiently as the sensor network changes.
- **Network Topology:** TinyDB manages the underlying radio network by tracking neighbors, maintaining routing tables, and ensuring that every mote in the network can efficiently and (relatively) reliably deliver its data to the user.

- **Multiple Queries:** TinyDB allows multiple queries to be run on the same set of motes at the same time. Queries can have different sample rates and access different sensor types, and TinyDB efficiently shares work between queries when possible.
- **Incremental Deployment via Query Sharing:** To expand the TinyDB sensor network, it is only required to simply download the standard TinyDB code to new motes, and TinyDB does the rest. TinyDB motes share queries with each other; when a mote hears a network message for a query that it is not yet running, it automatically asks the sender of that data for a copy of the query, and begins running it. No programming or configuration of the new motes is required beyond installing TinyDB.

2.2.3.1 TinyDB System Overview

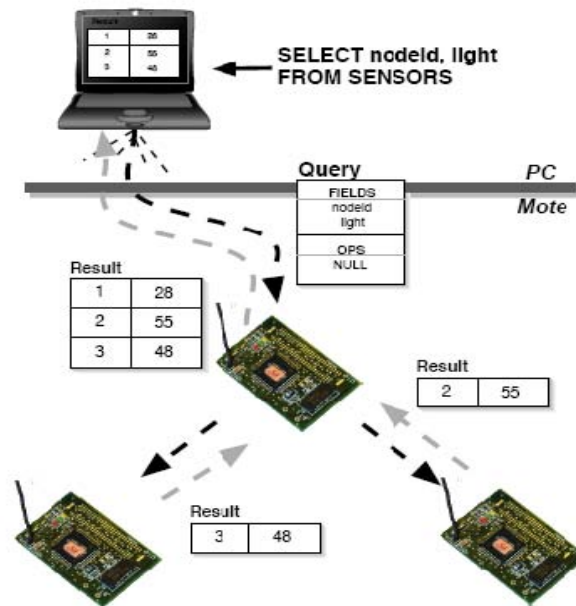


Figure 7 TinyDB

This section provides a high level overview of the architecture of the TinyDB software. It is designed to be accessible to users of the TinyDB system who are not interested in the technical details of the system's implementation.

We begin with a short description of a typical use-case for TinyDB. Imagine that John wants to find an unused conference room in his sensor equipped building, and that an application to perform this task has not already been built. The motes in John's building have a sensor board with light sensors and microphones and have been programmed with a room number. John decides that his application should declare a room in-use when the average light reading of all the sensors in a room is above a certain value. John wants his application to refresh this occupancy information every 10 minutes. Without TinyDB, John would have to write a couple hundred lines of custom C code to collect information from all the motes in a room, coordinate the readings across sensors, aggregate these readings together to compute the average light and volume, and then forward that information from within the sensor network to the PC where the application is running.

He would then have to download his compiled program to each of the motes in the room. Instead, if the motes in John's building are running TinyDB, he can simply compose the following SQL query to identify the rooms that are currently in-use:

```
SELECT roomno, AVG (light), AVG (volume)
FROM sensors
GROUP BY roomno
HAVING AVG (light) >20
EPOCH DURATION 5min
```

TinyDB translates this query into an efficient execution plan which delivers the set of occupied rooms every 5 minutes. John simply inputs this query into a GUI - he writes no C code and does not worry about how to install his code, how to propagate results across multiple network hops to the root of the network, how to power down sensors during the time when they are not collecting

and reporting data, and many other hard to do actions with sensor-network programming. The system can be broadly classified into two subsystems: Sensor Network Software: This is the heart of TinyDB, although most users of the system should never have to modify this code. It runs on each mote in the network, and consists of several major pieces [24]:

- **Sensor Catalog and Schema Manager:** The catalog is responsible for tracking the set of attributes, or types of readings (e.g. light, sound, voltage) and properties (e.g. network parent, node ID) available on each sensor. In general, this list is not identical for each sensor: networks may consist of heterogeneous collections of devices, and may be able to report different properties.
- **Query Processor:** The main component of TinyDB consists of a small query processor. The query processor uses the catalog to fetch the values of local attributes, receives sensor readings from neighboring nodes over the radio, combines and aggregates these values together, filters out undesired data, and outputs values to parents.
- **Memory Manager:** TinyDB extends TinyOS with a small, handle-based dynamic memory manager
- **Network Topology Manager:** TinyDB manages the connectivity of motes in the network, to efficiently route data and query sub-results through the network.
- **Java-based Client Interface:** A network of TinyDB motes is accessed from a connected PC through the TinyDB client interface, which consists of a set of Java classes and applications. These classes are all stored in the `tinyos-1.x/tools/java/tinyos/tinydb` package in the source tree. Major classes include:
 - A network interface class that allows applications to inject queries and listen for results
 - Classes to build and transmit queries

- A class to receive and parse query results
- A class to extract information about the attributes and capabilities of devices
- A GUI to construct queries
- A graph and table GUI to display individual sensor results
- A GUI to visualize dynamic network topologies
- An application that uses queries as an interface on top of a network of sensors.

2.3 Software Installation

This section will describe how to implement a Wireless Sensor Network using MICAz Motes loaded with TinyOS and TinyDB. We assume that the motes are not yet programmed with TinyDB. Once finished, the user will be able to use a Windows PC to insert SQL-like queries into the Sensor Network, perform some measurements and have the results returned to the base PC.

TinyOS is extremely sensitive to hardware and software configurations. The system has many dependencies that rely on specific versions of software components. Using older or newer versions of a particular component may cause incompatibilities resulting in the system refusing to build and execute [20, 22, and 23]. The kit includes a CD-ROM that contains all of the necessary software components to build a system from the ground up and allows you to perform some simple experiments.

2.3.1 Pc Requirements

We recommend running the software on a Windows XP service pack 2 box with a minimum of at least 1 GHz of CPU speed, at least 1 GB of RAM, one serial port, one Ethernet port, and have at least 2GB of free space.

The software will run at lower specs but we found that it operated much better with the previous specs if not higher. Of course, the more resources a Pc will have the better.

TinyOS and TinyDB have three major components.

1. A java based Graphical User Interface (GUI) that runs on a MS Windows.
2. A C flashed firmware that runs as an embedded application on the mote.
3. A software development environment that allows creation of the previous.

The hardware used in this guide was a selection of MICAZ Motes from Crossbow Technology Inc. The Motes were model MPR2400CA using the CC2420 2.4 GHz data radio. The sensor boards used were the MTS300CA which enables the mote to measure temperature, sound and light in addition to the battery voltage (used to power the Motes). The base station interface unit that we use to program the motes, model MIB510CA, is RS232 based and serves two main purposes:

- (1) It allows the user to reprogram any mote by plugging the mote directly into the base.
- (2) It operates as part of the root node interface giving the PC a data conduit onto the radio based sensor network.

For those who might use laptops a word of caution; The MIB510CA interface must be connected to the PC via RS232 which is a DE9 port (serial). Many modern laptops are being produced without this port. If your laptop does not have such a port, you must use a special cable called a “USB to Serial Adapter” which can be found at almost any computer hardware store (see figure below).



Figure 8 (DE9 port and USB cable)

2.3.1.1 Step one installing UCB TinyOS 1.1.11

The first step is to install the base system of TinyOS. All of the required files can be found on the included CDROM or downloaded from the following website:

<http://www.tinyos.net/dist-1.1.0/tinyos/windows/> .

We recommend downloading the installer from the website because the CDROM has an older version particularly TinyOS version 1.1.0-1is. The installer from the web page is a newer version called *tinyos-1.1.11-3is*. Please be aware that TinyOS v2.0 is out. The new version has more features but it still lacks to be a stable version. We will use 1.1.11.3is version because it is much more stable and plenty of support for it can be found online. Double click on the installer and please be patient while the system performs some system checks that can take up to 1 minute to complete; eventually the splash screen pictured above will appear. Unless you are an expert, pick complete when asked what type of installation you want.

Depending on the speed of the machine and whether or not anti-virus software is installed, it can take up to 15 minutes for the Installer to complete all steps. Please be aware that Anti-virus software will dramatically slow down the installation as it scans thousands of files, many of them java source files which tend to be targets of virus writers.

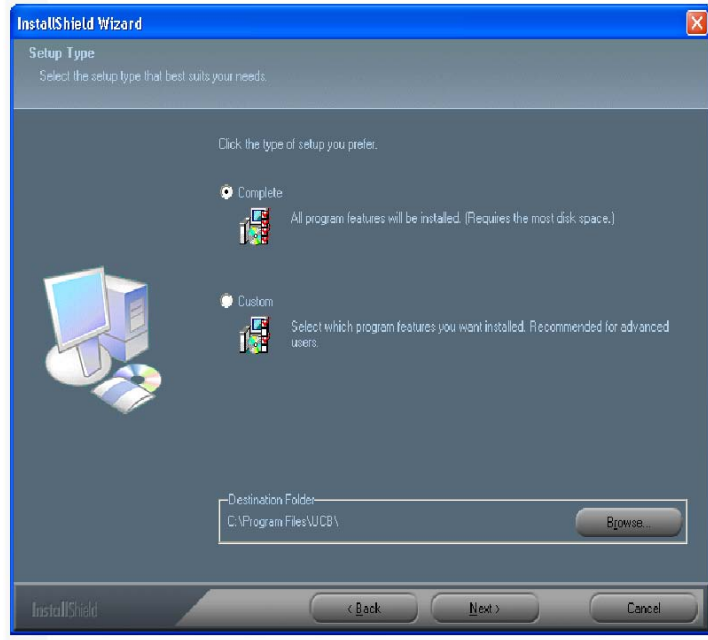


Figure 9 TinyOS set up splash screen

2.3.1.2 Step 2 Post Installation procedures

After the installer finishes you should now have a Windows Program Group called Cygwin along with a desk top short cut. Cygwin is free software that provides a Unix-like environment and software tool set to users of any modern version of MS-Windows for x86 CPUs (95/98/NT/2000/ME/XP). Cygwin consists of a UNIX system call emulation library, `cygwin1.dll`, together with a vast set of GNU and other free software applications organized into a large number of optional packages. Among these packages are high-quality compilers and other software development tools, a complete X11 development toolkit, GNU emacs, TeX and LaTeX, OpenSSH (client and server), and much more, including everything needed to compile and use PhysioToolkit software under MS-Windows[21].

Start the 'Cygwin bash shell' which will give you a Unix-like environment under Windows.

If you do not receive an error, skip down to the next section and go directly to 'Step 3 Updating TinyOS 1.1.15 cvs build'. The first time you start the bash shell, it is possible that an error will

be printed complaining about some missing security files. Not every installation of TinyOS will receive the error pictured below, it will depend on what networking protocols are configured into the PC used and if the users are authenticating against a network server (such as Microsoft Active Directory). As suggested by the error message pictured below, run the following two commands:

```
'mkpasswd -l -d > /etc/passwd '
```

```
'mkgroup -l -d > /etc/group '
```

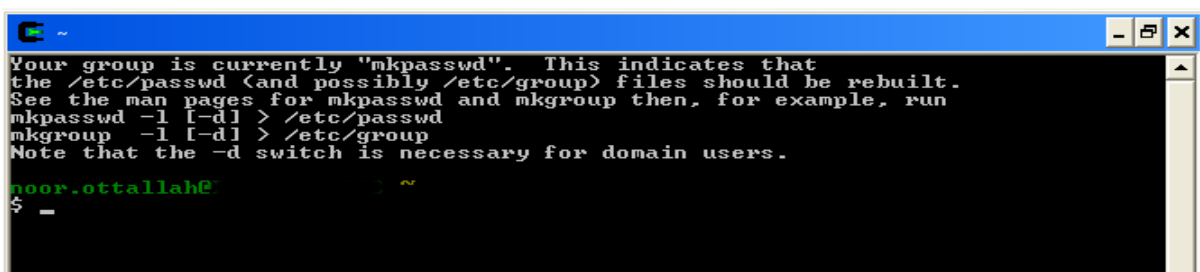


Figure 10 password error

After fixing the errors, close the shell and open a new one. A new different set of messages will pop up. The output should look similar to the figure below. Once again you should exit the shell. You have completed the initial installation of the base software.

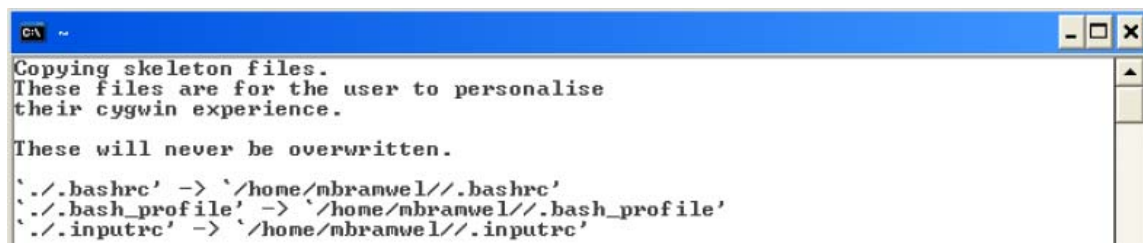


Figure 11 final message

2.3.1.3 Step 3 Updating to TinyOS 1.1.15 cvs build (optional)

Now that TinyOS 1.1.11 is installed, the system should be immediately upgraded to version 1.1.15. Open a new Cygwin shell window. You must change the working directory to the same directory that contains the updated RPM file. RPM is an abbreviation that stands for *RPM Package Manager*. RPM is also the name of a program that enables the installation, upgrading and removal of packages. TinyOS is updated through RPMs which you download from the website or copy from the installation CD. We recommend getting the latest version off the TinyOS website. If you use the CD option then you will need to mount the drive and access the folder. To change the working directory to the software directory which in our case, the CDROM was Windows drive F: which was mounted as `‘/cygdrive/f’` we enter the following:

```
‘cd /cygdrive/f/software’
```

We then install the updated system package by entering the following command:

```
‘rpm --force -ivh --ignoreos *.rpm’
```

If you download the package from the web you only need to change the working directory to the folder where the RPM was downloaded and basically run the previous command.

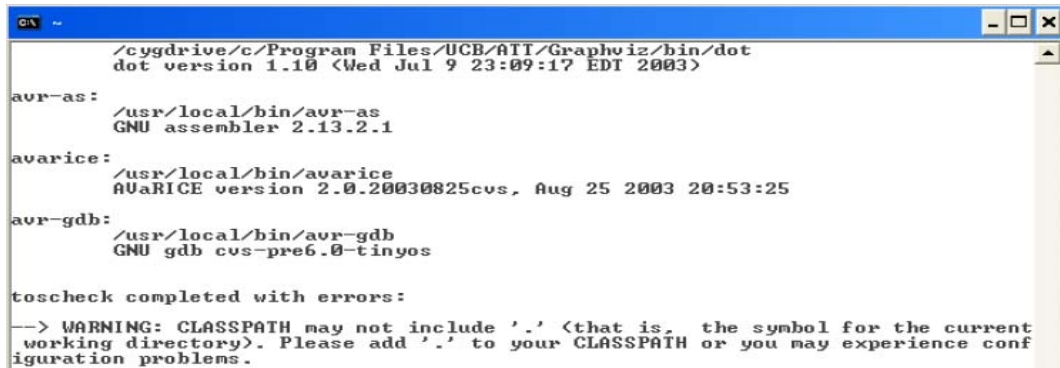
2.3.1.4 Step 4 Checking the TOS Installation and CLASSPATH

To ensure that the system is now configured properly and has all the packages and libraries, the TinyOS distribution includes a program that will verify the configuration of your machine and will report back if any errors are found. The test must finish without any errors. Do not continue to any other steps if you have errors in your configuration.

To test your software configuration, start a 'Cygwin shell' and type the following command:

`'toscheck'`

Most likely an error complaining about the CLASSPATH will pop up on the screen.. The CLASSPATH error is caused by a configuration issue with the java environment and will look similar to the screen shot pictured below.



```
dot version 1.10 (Wed Jul 9 23:09:17 EDT 2003)
avr-as:
/usr/local/bin/avr-as
GNU assembler 2.13.2.1
avarice:
/usr/local/bin/avarice
AVaRICE version 2.0.20030825cvs, Aug 25 2003 20:53:25
avr-gdb:
/usr/local/bin/avr-gdb
GNU gdb cvs-pre6.0-tinyos

toscheck completed with errors:
--> WARNING: CLASSPATH may not include '.' (that is, the symbol for the current
working directory). Please add '.' to your CLASSPATH or you may experience conf
iguration problems.
```

Figure 12 CLASSPATH error

To fix this problem, a configuration file needs to be open to make a few needed changes. Open any preferred editor and edit the file named 'javapath' located at

`'/opt/tinyos-1.x/tools/java/javapath'`

Browse down to line 41 in the file where it shows this:

```
unshift @add, ".";
```

You should change it to this:

```
#unshift @add, ".";
```

Browse down to lines 57 & 60 where they show this:

```
print "$addpath;$oldpath\n";
```

You should change it to this:

```
print "$addpath;$oldpath;\n";
```

Browse to Line 64 where it shows this:

```
print "$addpath\n";
```

You should change it to be this:

```
print "$addpath;.\n";
```

After making the necessary edits to the file, save and exit the shell. Open a new shell and re-run the *toscheck*. It should now complete without errors.

2.3.1.5 Step 5 Building the java tools

We need to run a couple of scripts that will tell the system to grab all the information needed to compile java classes. To start, type in the commands below and wait for them to complete this will compile the java tools on your system:

```
cd /opt/tinyos-1.x/tools/java make
```

```
cd /opt/tinyos-1.x/tools/java/jni make install
```

2.3.1.6 Step 6 Running the TinyDB GUI for the PC

The TinyOS and TinyDB system is comprised of 2 major components. One part of the system runs as a java application on the ROOT PC providing the menus and graphing environment; the other part of the system runs as compiled C code on the motes themselves. To compile the java GUI for the PC, type the following commands:

```
‘ cd /opt/tinyos-1.x/tools/java/net/tinyos/tinydb/ make’
```

The system might take a while to finish depending on the speed of the machine during which a lot of activity will show on the screen. If at the end you see an error it will most likely be because one of the following:

- 1-A java compile error; you are not in the right directory to compile from.
- 2- Javax not found. This is more likely to happen please check the env PATH and make sure the system has the right java path included. There is a java.exe in the Windows folder that Cygwin looks at first. Either delete it from the path or add the correct path of the real java path in front of it. See the figure below:

```

noor.ottallah@...
$ echo $path

noor.ottallah@...
$ echo $PATH
/cygdrive/c/tiny/jdk1.4.1_02/j2sdk1.4.1_0
/X11R6/bin:/cygdrive/c/Perl/bin:/cygdriv
/cygdrive/d/Oracle/OraDS_1/jdk/jre/bin:/c
ient:/cygdrive/d/Oracle/OraDS_1/jlib:/cyg
Oracle/OraDS_1/jre/1.4.2/bin/client:/cygd
gdrive/c/WINDOWS/system32:/cygdrive/c/WIN
/cygdrive/c/Program Files/Reflection:/cy
op Validator:/cygdrive/c/Program Files/F
rogram Files/SSH Communications Security/
iles/Diskeeper Corporation/Diskeeper:/cy
/bin:/cygdrive/c/tiny/ATI/Graphviz/bin:/
stem:/cygdrive/d/Documents and Settings/
r/local/mspgcc/bin
noor.ottallah@...
$

```

Figure 13 Java path

The following commands will start the TinyDB GUI on the PC:

```

cd /opt/tinyos-1.x/tools/java

java net.tinyos.tinydb.TinyDBMain

```

Please be aware of the following:

- 1- The commands listed above are case sensitive. They must be entered exactly as shown otherwise a 'java.lang.NoClassDefFoundError' message will be displayed and the system will refuse to start.
- 2- If you are using anything but COM1, you will receive an error when the GUI tries to start. The resolution is to modify the TinyDB configuration file to point to the proper COM port.

As an example, on one particular machine, Windows XP assigned COM3 to the device.

The reference from COM1 had to be changed to COM3 for the hardware to talk to the machine and display results. The filename that required modification was:

```
‘/opt/tinyos-1.x/tools/java/net/tinyos/tinydb/tinydb.conf’
```

Line 28 used to have this:

```
comm.string:serial@COM1$57600
```

We changed line 28 now has this:

```
comm.string:serial@COM3$57600
```

After the modification was completed, java was able to start TinyDB without difficulty.

2.3.2 Installation of apps on the Nodes

This section of the document will describe the steps required to setup, test and program the MICAz hardware with the appropriate codes. The MICAz platform has a single slide switch. It is the power switch for the batteries. We mentioned earlier that it is very important to make sure the switch is in the off position. This will prevent the mote from burning out. With that done, it is unnecessary to remove the batteries if you ensure that the slide switch is always in the OFF position whenever the mote is plugged into the MIB510CA programmer by connecting the daughter port of the mote into the daughter port of the MIB510CA. There is no wrong way to plug them both together where one side is female and the other is male. The programmer has a single slide switch that affects how data is sent on the serial com port. It must be in the OFF position to both program and use the motes.

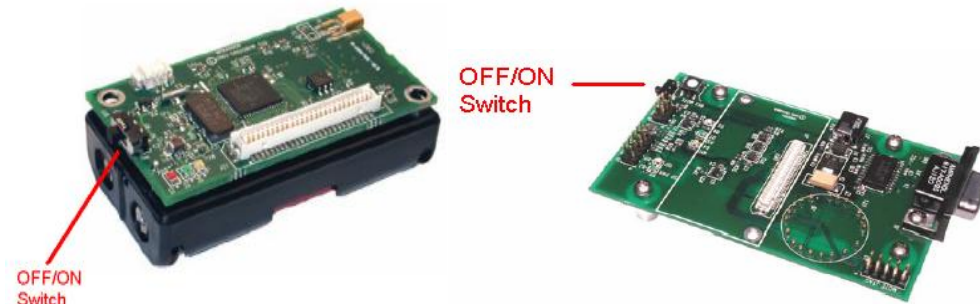


Figure 14 On switch on mote and programmer

To test the hardware and get some sensor results we need to do the following:

1. Connect the hardware
2. Compile and load a program onto the MICAz
3. Run an application on the PC that communicates with the MICAz

2.3.3 Connecting the hardware

To set up the hardware please insure the following steps are done:

- 1- You must be connected to the PC with a serial cable or USB cable.
- 2- We will assume in this guide you are connected to COM1.
- 3- You must plug a MICAz into the MIB510CA programmer.
- 4- A sensor-board is not required for this test.
- 5- The programmer must be powered via a wall adapter
- 6- Both the MICAz and the MIB510CA must have their slide switches set to OFF.



Figure 15 Complete setup

2.3.3.1 Installing a simple app 'MicaHWVerify'

Open a 'Cygwin bash shell' and type the following commands:

```
'cd /opt/tinyos-1.x/apps/MicaHWVerify make micaz'
```

At this point you now have a custom MICAz compatible program ready to be programmed into the mote. The previous step created an application ready for transferring into the MICAz. To write the firmware image to the MICAz chip, type the following command:

```
'make micaz reinstall mib510,/dev/ttyS0'
```

Please be aware again that the above command is case sensitive. Also, be aware of the COM port the hardware is connected to:

```
/dev/ttyS0 = com1 /dev/ttyS1=com2 /dev/ttyS2=com3 /dev/ttyS3=com4
```

The screen will show some info about the programmer chip and if all goes well a message with the word 'fuse'x09 or similar will show. You have successfully uploaded the image to the mote.

Next, you will attempt to communicate with the MICAz. To create the PC test program, type the following command:

```
make -f jmakefile
```

To run the program and display a small report, type the following command:

```
'MOTECOM=serial@COM1:micaz java hardware_check'
```

If all goes well, you should see a small report that displays the Node Serial ID (Figure 16)



```
~/opt/tinyos-1.x/apps/MicaHWVerify
$ make -f _jmakefile
nig java -java-classname=DiagMsg MicaHWVerify.nc DiagMsg -o DiagMsg.java
struct DiagMsg 17 18
  serialId (8)U 0 8
  flashCheck (3)U 64 8
  SPIFix U 88 8
  flashConn U 96 8
  rxTest U 104 16
  count U 128 8
  sendType U 128 8
nig java -java-classname=RxTestMsg MicaHWVerify.nc RxTestMsg -o RxTestMsg.java
struct RxTestMsg 2 32
  value U 0 16
javac -sourcepath . hardware_check.java
~/opt/tinyos-1.x/apps/MicaHWVerify
$ MOTECOM=serial@COM1:micaz java hardware_check
Hardware check started
serial@COM1:57688: resynchronizing
Hardware verification successful.
Node Serial ID: 1 45 74 b3 a 0 0 a2
~/opt/tinyos-1.x/apps/MicaHWVerify
```

Figure 16 Hardware test

2.3.3.2 Testing the radio functionality

This test requires two MICAz nodes. One unit will serve as the root node and the second node will be used as the remote node. During the test, the 2 nodes will communicate with each other via the radio link.

Node #1:

- Take one of the nodes that has passed the 'Hardware Comms Check', install batteries and slide the switch to ON.
- The lights on the MICAz will start to blink, the node is now active!

Node #0:

- This node will be our root node and will stay connected to the programmer board.

It must have some special 'base station' software programmed onto the MICAz to work.

To compile and load the base station software onto the MICAz, type the following commands:

```
'cd /opt/tinyos-1.x/apps/TOSBase make micaz'
```

```
'make micaz reinstall mib510,/dev/ttyS0'
```

Re-run the 'Hardware Comms Check'

```
'cd /opt/tinyos-1.x/apps/MicaHWVerify'
```

```
'MOTECOM=serial@COM1:micaz java hardware_check'
```

The Node Serial ID report will be displayed. Notice that the ID of the remote node was displayed, not the Node ID of the MICAz plugged directly into the base station. The base station acted as a data conduit passing the serial data onto the radio network; the remote node saw the data on the RF interface and responded. TinyOS ships with plenty of already made apps which are located under: '/opt/tinyos-1.x/apps/'. Feel free to look at them all as some might come in handy.

2.4 Installing TinyDB application

Each MICAz must be programmed with a unique copy of the TinyDB software. One node is called the ROOT Node and during normal operation is always connected directly to the base station (MIB510CA programmer board). The Node ID of the ROOT Node is always zero. All other nodes must have a unique non-zero Node ID. The following steps will compile and install TinyDB on each MICAz node.

1. Change the working directory to the TinyDB directory:

```
'cd /opt/tinyos-1.x/apps/TinyDBApp'
```

2. Compile and install a custom version of TinyDB for the ROOT node (Node#0)

```
'make micaz install.0 mib510,/dev/ttyS0'
```

3. Remove the ROOT Node and connect one of your remote nodes. The following command sets the Node ID to “1” and programs the MICAz node:

```
‘make micaz reinstall.1 mib510,/dev/ttyS0’
```

Repeat Step #3 until all MICAz nodes have been programmed. Remember to increment the Node ID giving each node a unique number. You are now ready to run the TinyDB application on the motes and PC.

2.4.1 Starting the TinyDB GUI on the PC

The following commands will start the TinyDB GUI on the PC:

```
‘cd /opt/tinyos-1.x/tools/java’
```

```
‘java net.tinyos.tinydb.TinyDBMain’
```

If the system refuses to start, refer to step 6 in this document for more information.

2.4.2 Displaying the Topology

The first test of the Sensor Network is to determine if indeed you have a network. When you start the TinyDB GUI, a screen similar to the picture below will appear. Clicking on the [Display Topology] button causes the system to insert a query into the network that causes all nodes to respond. Their responses prove that each node is connected and alive. Have patience when you attempt to display the topology. It usually takes about 4-30 seconds to see some action. The delay depends on how many nodes are in the network.

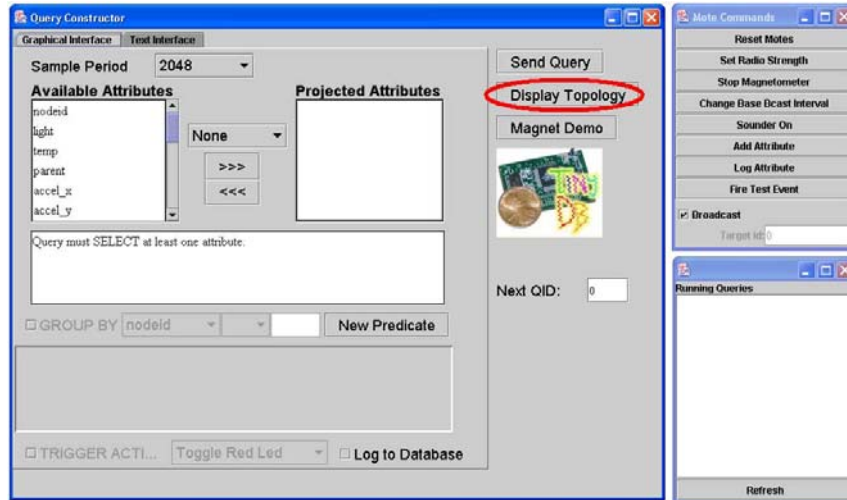


Figure 17 TinyDB main pages

The ‘Sensor Network Topology’ screen pictured below is a graphical representation of the layout of the sensor network. It does not however reflect the actual locations of the sensors. The ‘Light’ attribute is used to provide a visual indication of the attributes value. Node 0 is the ROOT Node and was plugged directly into the base station interface. All other nodes were scattered around three adjoining offices. The office containing Node#4 had the lights turned off (the room was dark) and the graph clearly shows a low value for that attribute. Node#2 was on the floor behind a desk in a shadow; the graph shows a darker (but not dark) region near that node. All other nodes were in brightly lit areas.

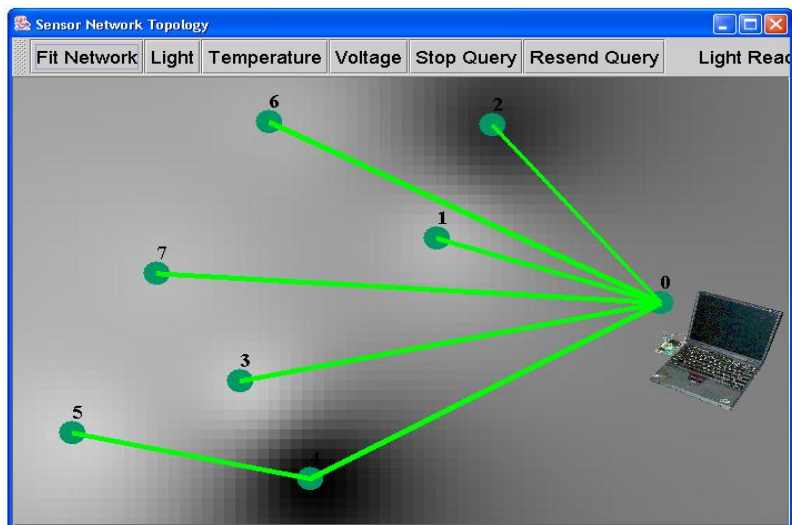


FIGURE 18 SENSOR TPOLOGY AND READINGS

2.4.3 Simple Queries

The strength of the TinyDB GUI is the ease in which the user can inject queries into the Sensor Network and have the results returned and displayed on the base station PC. Pictured below is the main screen of TinyDB. To start a query, scroll through the list of available attributes in box #1. Note that although the list contains all supported attributes, not all nodes will have the sensors installed to perform that measurement. Make sure you do not try to read from something that does not exist. When you double-click on an attribute in box #1, the attribute will appear in the listing in box #2 as well as a SQL-like query will be constructed in box #3. If you have clicked on something accidentally, double-click on the attribute in box #2 to remove it from the list. When you have selected all of your desired attributes, click on #4 [Send Query] to start the querying of the network.

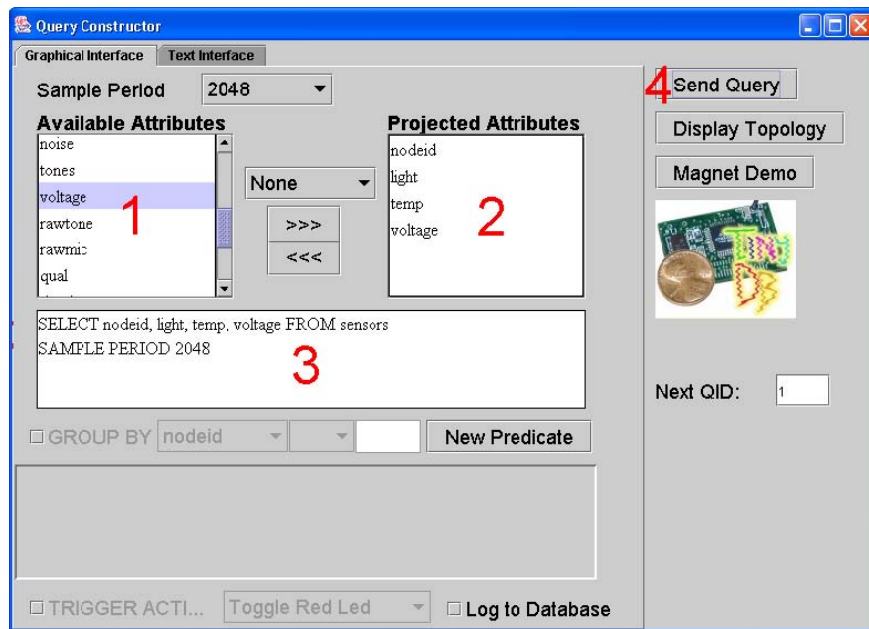


Figure 19 GUI sections

Once the query has been injected in the Sensor Network, the nodes will perform readings and send back the results to the base station PC connected to Node 0. A chart will be created in real-time that looks similar to the one pictured below.

2.4.4 Results being received and charted by TinyDB

The above graph represents the plot of light from 3 nodes over a period of 2,490 samples. Have patience when you send a new query into the network. It can take over a minute for the query to be encoded, and transmitted to all the nodes for the first response to migrate through the network back to the base station PC. Once the results start to arrive, they will do so at the expected rate without delay.

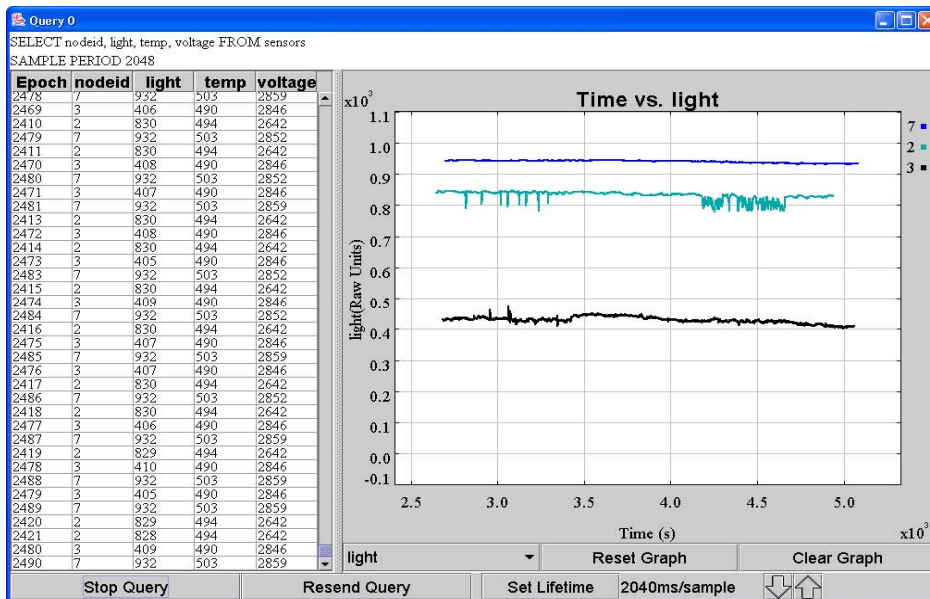


Figure 20 Graph actions

The TinyDB GUI provides a graphing interface but it is not the best method of displaying the results from multiple attributes. The best way is to collect the data and read into Microsoft Excel and create a graph.

2.5 Storing the Results Data in PostgreSQL (Optional)

When TinyDB receives results from the Sensor Network, the data is stored and displayed within the TinyDB GUI as a spreadsheet. Although this is adequate for quick adhoc queries, it becomes difficult to access the data if queries are performed over an extended period of time. It may be desirable to have the data stored within a PostgreSQL database [25].

PostgreSQL is an open source project offered as a free download from <http://www.postgresql.org/>. PostgreSQL is available on a wide variety of platforms including but not limited to Windows, Macintosh/OSX and Linux. It is suggested that the above website is used to download and install the software on the server platform of your choice. If you decide to use the Windows version of PostgreSQL, a full-screen graphical interface is provided in addition to the command line tools.

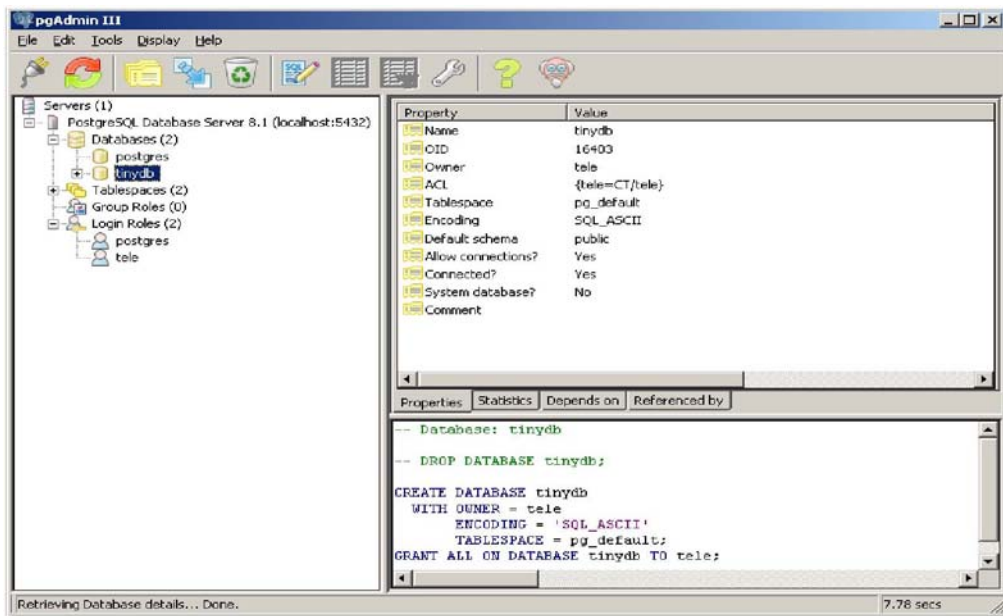


Figure 21 PostgreSQL

To setup PostgreSQL you need to do the following:

- 1-TinyDB requires a database to be created called 'tinydb'.
- 2-A username called 'tele' must be created and requires permission to access 'tinydb'.
- 3-The PostgreSQL command to create the table is:


```
'create table queries (query_table varchar(10), query_time timestamp, query_string varchar(500));'
```
- 4-The TinyDB configuration file must be edited to reflect the username/password and location of the database.

The configuration file can be found in the following location:

`/opt/tinyos-1.x/tools/java/net/tinyos/tinydb/tinydb.conf`

Postgres server settings:

- `postgres-user:tele`
- `postgres-passwd:tiny`
- `postgres-db:task`
- `postgres-host:localhost`

Once the changes have been made to reflect the true location and attributes of the PostgreSQL database, you will be able to use the [Log to Database] option in TinyDB.

Chapter 3 System Implementation

Until now there has never been an effort to create a complete system that records sensor readings from the surroundings and at the same time be able to manage security keys that nodes use to establish secure connections. We imagined a system that once activated; the nodes will automatically set themselves up by finding their neighbors, establish secure connections even if the neighboring do not share the same keys, and finally begin to report their sensor readings back to the base station.

Imagine the military sending thousands of sensor nodes on a plane to get dropped in a hostile area the size of 50 football fields. The sensors will report back with diverse measurements like the presence of toxic gas, infantry locations, and snap shots of the area at certain times. It is critical in such scenario that all the readings sent back to the base be encrypted. It will be most likely that the nodes will be prepackaged with the keys before deployment using the techniques discussed in [12, 13, 15, 16, 18, and 26]. The probability of all the nodes establishing secure connections will be very high using the previous algorithms especially when the topology is known [27, 28]. Still, there will always be a probability that some nodes will not be able to establish secure connections because some nodes will land next to other nodes that do not share the same key. The topology is never guaranteed when dropping from a drone or an F-15 combat fighter. Now assume the nodes that could not establish a connection are in critical territories. The whole deployment operation would be considered a failure. Millions of dollars would have been spent for nothing and lots of lives taken as well. To fix the problem, the unconnected nodes would have to be retrieved, reprogrammed and redistributed once more. In this situation this would be impossible.

In comes our system. Our system builds on top the functionality of the TinyDB application. TinyDB (discussed earlier in chapter 3) is a query processing system for extracting information from a network of sensor nodes. Unlike other solutions for data processing in TinyOS, TinyDB does not require you to write embedded C code for sensors. Instead, TinyDB provides a simple, SQL-like interface to specify the data you want to extract, along with additional parameters, like the rate at which data should be refreshed -- much as you would pose queries against a traditional database. Given a query specifying your data interests, TinyDB collects that data from motes in the environment, filters it, aggregates it together, and routes it out to a PC or base station. TinyDB does this via power-efficient in-network processing algorithms. TinyDB is a well tested and proved to work quiet well in all conditions [29].

Basically our system which we will call TinyDB+ from here after will be able to display, along with the normal sensor readings, the security keys a node is using to establish a connection. This new function will be integrated into the GUI for easy retrieval and changing. We assume as a reason to pursue this system is that the sensor nodes will be deployed randomly. We also assume that the nodes will use keys that are loaded by using the pre-distribution key scheme.

3.1 Design

In the previous example where the nodes were dropped in a hostile area, some nodes landed next to nodes that did not hold the same secret key. This resulted in the nodes not creating secure paths rendering the nodes useless to the whole network. Now using our system; if the nodes were programmed with TinyDB+ the previous situation could have been fixed easily by doing the following:

- 1- Query the network for node connectivity using the GUI which should display a simple graph representing the connected nodes. Nodes that have links (connecting lines) means that they have created secure routes. Nodes that are lonely mean that they could not find other nodes in their range that share the same key.
- 2- Query the lonely nodes for their key that is built in to them.
- 3- Query the closest nodes to the lonely node for their key.
- 4- Using TinyDB+ we can re-key the lonely mote using Over the Air programming making sure that the new key will be compatible with that of the lonely node's neighbor.
- 5- Restart the newly keyed node and enjoy a fully connected network.

The following diagram should give a better understanding:

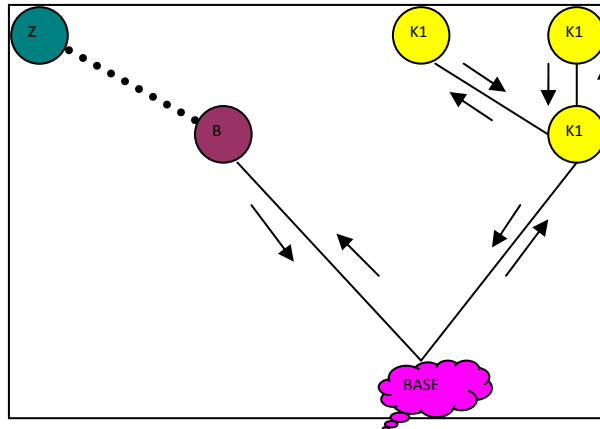


Figure 22 (a) nodes Z and B don't share the same key

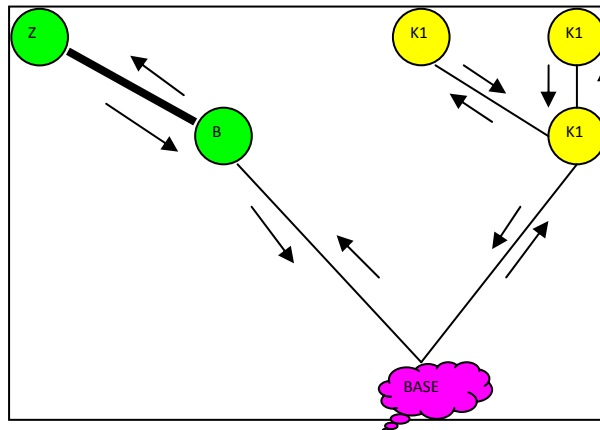


Figure 23 (b) nodes Z and B share the same key the system is fully connected

The system needs to be small enough to still fit on the node and all of TinyDB's original functionality should be untouched unless enhanced. The new features should have the same operating threshold the older ones have. In other words, the new features should not crash the system nor exhaust the nodes resources in any different way the older features do. TinyDB+ will also be secure when handling the key management scheme.

3.2 Implementation

In order for us to integrate the secret Key functionality into TinyDB and create TinyDB+ we had to rewrite code in two main areas:

- 1- The TinyOS side of the TinyDB application that runs on the nodes.
- 2- The Java based application side (The PC side)

3.2.1 TinyOS changes

TinyDB relies on a collection of TinyOS components called Tiny Schema [30] that manages a small repository of named attributes, commands and events that can be easily queried, invoked or signaled from inside or outside a sensor network. A Tiny Schema *attribute* is much like a column in a traditional database system. It has a name and a type. In addition, Tiny Schema allows you to associate arbitrary TinyOS code to each attribute for getting and setting the attribute value. Once an attribute is created, it can be retrieved or updated through a unified interface provided by Tiny Schema. There are three types of attributes that Tiny Schema provides:

- **Sensor Attributes.** These can be raw readings from sensors such as temperature and photo sensors, accelerometers, magnetometers, etc. They can also be computed sensor values after applying some calibration or signal processing logic.
- **Introspective Attributes.** These are values from internal software or hardware states, e.g., software version stamp, parent node in routing tree, battery voltage, etc. They are very useful for monitoring the health and statistics of a mote network.
- **Constant Attributes.** These are constant values assigned to a mote at programming time or run time, e.g., node id, group id, name, location, etc.

In order for TinyDB+ to retrieve secure keys we introduced a new constant attribute into the system. Currently all attributes must be statically built into each mote. To be able to see the key and change it we need to rely on another system aspect of Tiny Schema. We use the Tiny Schema *command* which is much like a stored procedure in a traditional database system. It consists of a name, a list of arguments and a return type. You can associate arbitrary TinyOS code to each command. Tiny Schema provides a unified interface for invoking these commands. TinyDB is also built on top of the Tiny Schema command interfaces for its trigger actions [21, 30]. Typically, there are two classes of commands:

- Actuation Command. These are commands that cause some physical actions on a mote, e.g., rebooting a mote, flash LEDs, sound buzzer, raise a blind (when connected to an appropriate actuator), etc.
- Tuning Command. These are commands that adjust internal parameters, e.g., routing policy, number of retransmissions, sample rate, etc.

We also rely on the Tiny Schema *event* which is introduced to capture asynchronous events in sensor networks, e.g., detection of a car, push of a button, etc. Tiny Schema provides interfaces for registering and invoking events as well as associating Tiny Schema commands with events as callbacks when events are signaled. At the moment, all commands also must be statically built into the node before deployment. Attributes are stored in the `tinyos-1.x/tos/lib/Attributes/` folder under the TinyOS installation directory. We created two new files in this directory to create our new attribute; the `Attrkey.nc` file and the `AttrkeyM.nc` file.

Essentially, the first file is the configuration file which basically defines the attribute and gives it a name and tells the system where to look for the full definition. The second file is the module hence the 'M' at the end of the name. The module file is where the full implementation of the commands and events are coded. The following gives an example of how an attribute is registered:

```
command result_t registerAttr(char *name, TOSType attrType, uint8_t attrLen)
```

The attrLen argument is only relevant to variable-length types such as STRING. It is ignored for fixed-length types.

```
event result_t getAttr(char *name, char *resultBuf, SchemaErrorNo *errorNo)
```

This is the TinyOS code that you must provide for getting the value of the attribute you just registered through registerAttr. The 'name' is the name of the attribute. It is mostly redundant, but may come in handy if you want to write one piece of code that supports multiple attributes. ResultBuf is a pointer to a pre-allocated buffer to hold the value of this attribute.

Next we needed to introduce the secret key as a global variable and build it in the system code before compilation. We did this by modifying the 'tos.h' file under the tinyos-1.x/tos/system folder. We declared a new variable called TOS_LOCAL_KEY. This is the variable that will hold the key value and will be initialized at compilation time. For starters we assume that during the key-distribution phase the nodes only get one key at first from the pool of keys .the new key will be set to the TOS_LOCAL_KEY variable. After deployment the nodes automatically figure out if they have matching keys with their neighbors or not. If they do not then we can query the surrounding nodes for their key and from their upload the new key to the

disconnected node. Finally we had to implement the new attribute into the ‘tinyDBAttr.nc’ file for the application to initialize the variable when started. This file can be located in tinyos-1.x/tos/lib/tinydb folder. Here is section from the file:

```
“includes Attr;

configuration TinyDBAttr {

    provides interface AttrUse;

    provides interface StdControl;

}

implementation {

    components Attr, AttrPot, AttrGlobal,AttrKey, TinyDBAttrM, TupleRouterM, AttrTime,
    TinyAlloc, NETWORK_MODULE....
```

3.2.2 Java Application changes

In order for the TinyDB GUI to display the new attribute we had to implement new features into the application code which is located in the ‘/opt/tools/java/net/tinyos/tinydb/’ folder. We modified the tinyos-1.x/tools/java/net/tinyos/tinydb/catalog.xml file to include information about your new attribute in the java GUI. You will need to include an <attribute> record describing your component to the <attributes> element. The basic structure of these elements is as follows:

```
<attribute>
<name> </name>
<type> </type>
<minval> </minval>
<maxval> </maxval>
<isConstant> </isConstant>
<joulesPerSample> </joulesPerSample>
</attribute>
```

The last four elements are optional. Name is the name for the attribute that should be used in the query and will be shown in the GUI. Type is the data type of the attribute; it should be one of int8, uint8, int16, uint16, int32, uint32, or string. Minval and maxval are used in the visualization and for query optimization. isConstant is a Boolean specifying whether the attribute is a constant value; this will eventually be used in query optimization but is currently unused.

‘JoulesPerSample’ is an approximation of the energy required to acquire a sample from this attribute; this is also used in query optimization [22, 29]. We added the new definition of a ‘node key’ into the file and gave it all the crucial values.

3.3 Securing TinyDB+

Up to now all the implementation we did so far was to introduce the notation of a key in to the system. We still needed the TinyDB application to use secure connections because by default it does not. TinyDB does not use keys as a way to encrypt the sensor readings when sending back to the base. To make it use secure key transmission we had to wire the TinySec component in to the TinyDB application that resided on the nodes.

TinySec [31] is an efficient block cipher and keying mechanism that is tightly coupled with the Berkeley TinyOS radio stack. Before transmitting a packet, each node first encrypts the data and applies a Message Authentication Code (MAC), a cryptographically strong extremely hard to forge hash to protect data integrity. The receiver verifies that the packet was not modified in transit using the MAC and then deciphers the message.

Enabling TinySec was a straightforward process. In general, the application code did not need to be changed. We only needed to add `TINYSEC=true` to the application's Makefile in order to enable TinySec. By default, TinySec will authenticate all messages, but encryption is turned off. To enable key encryption we had to be aware of the keys and the key-file. Each Mica mote can only communicate with other motes that have been programmed with the same key or have at least one matching key. The keys are currently set in a given program at build time.

3.4 Changing the Keys in TinyDB+

There are two ways we can change the keys on a node if it so happens that it does not share a common key with one of its neighbors:

- 1- Using TinySec built in methods that are automatically added to TinyDB+.
- 2- Using over the Air reprogramming.

3.4.1 Key updating using TinySec

The TinySecControl interface exported by the TinySecC component enables you update TinySec keys and query and reset the initialization vector (IV). The TinySecControl interface has six commands:

- `command result_t updateMACKey(uint8_t * MACKey);`
- `command result_t getMACKey(uint8_t * result);`
- `command result_t updateEncryptionKey(uint8_t * encryptionKey);`
- `command result_t getEncryptionKey(uint8_t * result);`
- `command result_t resetIV();`
- `command result_t getIV(uint8_t * result)`

The `updateMACKey ()` and `updateEncryptionKey ()` update the corresponding key, and take a pointer to an array of TINYSEC KEYSIZE bytes. These commands return SUCCESS if the key update is successful. These commands will return FAIL if the key update is attempted while any crypto operation is executing in TinySec. This is the simplest solution to ensure atomicity of crypto operations with respect to the TinySec keys [31, 32]. The caller of these commands must check their return values, and try again if they return FAIL. The `getMackey ()` and `getEncryptionKey ()` return the corresponding key into the array referenced by the input parameter result. Result must reference a buffer of size at least TINYSEC KEYSIZE bytes.

The `resetIV ()` resets the counter portion of the IV to 0. This will most likely be used in concurrence with updating the encryption key. As in the key update commands, `resetIV ()` will return FAIL if it is called while any crypto operation is executing in TinySec. The `getIV ()` returns the current value of the IV into the array referenced by the input parameter `result`. `Result` must reference a buffer of size at least `TINYSEC IV LENGTH` bytes [32].

This type of update can only be used to change connected nodes and not lonely ones. So just in case an attacker was to compromise a node and retrieve its keys we can use this type of update to change the keys in the network with out bringing all the nodes down. We did not test this as we were more concerned in changing the keys that were carried by lonely nodes. This option will be enhanced in the next version.

3.4.2 Key updating using Over the Air reprogramming

Network programming or over-the-air programming is very important function for wireless sensor networks (WSN) [33]. Deluge [34], Infuse [35] and Multi-hop Network Programming [36] are some examples of network programming protocol .They are used to update sensor nodes over a wireless connection. In network programming, the program binary image is generally fragmented to be transported from base station to sensor nodes. Deluge, Infuse and Multi-hop Network Programming are some examples of the protocol.

TinyOS network programming consists of three main components: Deluge, NetProg, and TOSBoot [21]. Deluge is a reliable data dissemination protocol for disseminating large binary objects to all nodes over a multihop wireless network. The Deluge component is used to inject a new program image for execution on TinyOS nodes.

The NetProg component provides the crucial functionality for saving TinyOS state about the node before reboot, invoking TOSBoot, and restoring state at boot time.

TOSBoot is responsible for programming the micro-controller and is a static piece of code that is guaranteed to execute when the micro-controller exits the reset state.

Basically the way this process works, is when a lonely node is found we figure out its neighbors by using the GUI .TinyDB+ can report back with the neighbor's key or, if we want, we can make it return a hash number in place of the KEY pool it has on board. This is a much secure approach because an attacker can not figure out what the key is. Then, we compile a new image of the TinyDB application which has a new KEY pool set or at least one common key with a surrounding neighbor. Then, the host program reads the application program code and prepares code packets to disseminate. In the second step, the host program sends the code packets. And the sensor nodes store the code packets in the storage space after receiving them. Finally, the network programming module rebuilds the program code and calls the boot loader to transfer the program code to the program memory.

This is where Deluge needed to be modified. Deluge is considered to be an epidemic protocol meaning that once an image is pushed all the nodes will ultimately reprogram themselves [34]. Our mission then was to fix the Deluge protocol to accept 'node Id' as a parameter that constituted which node should reprogram itself with the new image. We also had to add a new check method in the boot loader that checks the 'node Id' in the image with the 'node Id' of the node it resides on. Now, even thou all the nodes get the new image they will not reboot. Only the intended recipient will reprogram and reboot with the new image.

Chapter 4 Tests and Results

We put TinyDB+ to work to test its functionality. We divided the test procedures into steps to test the system from the ground up. The first step was to validate that the new image, created after modifying TinyDB, was still small enough to fit the nodes. The TinyDB app before modification memory foot was:

61716 bytes in ROM

2825 bytes in RAM

The RAM foot print for the application should not surpass 3100 Bytes .TinyDB+ memory footprint was:

61762 bytes in ROM

3073 bytes in RAM

TinyDB+ is small enough to fit on a node and still leaves enough room for extension for future enhancements.

4.1 Connectivity

In the second step we first established that the addition of the new attribute (NODE KEY) functions properly. This test by its self was tested in four different stages; the first stage showed that GUI loads and that the new attribute displays accurately with in it. The second stage was to activate the system with one node that did not carry any secret keys. In our system we assume that a node with no key installed reports back with a zero. The third stage was to add one key to the same node and query the system for the new key.

The final stage was to add more than one node with different keys and request the system to display their keys. We like to make a note of that for this step TinySec was turned off. We found for all the previous stages the system worked flawlessly and did what it needed to do. The following pictures are the proof:

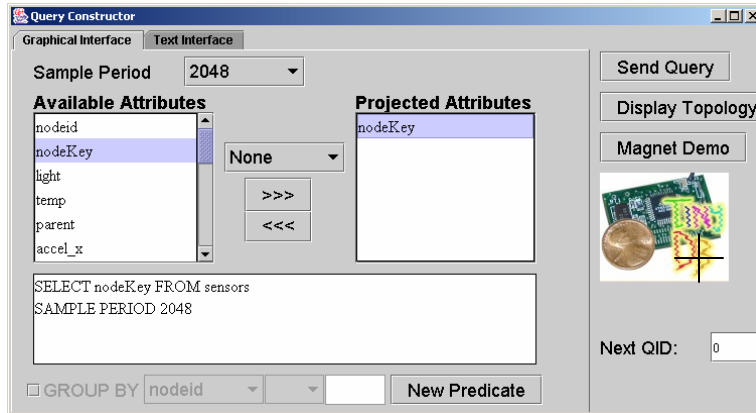


Figure 24 The new TinyDB+ interface.

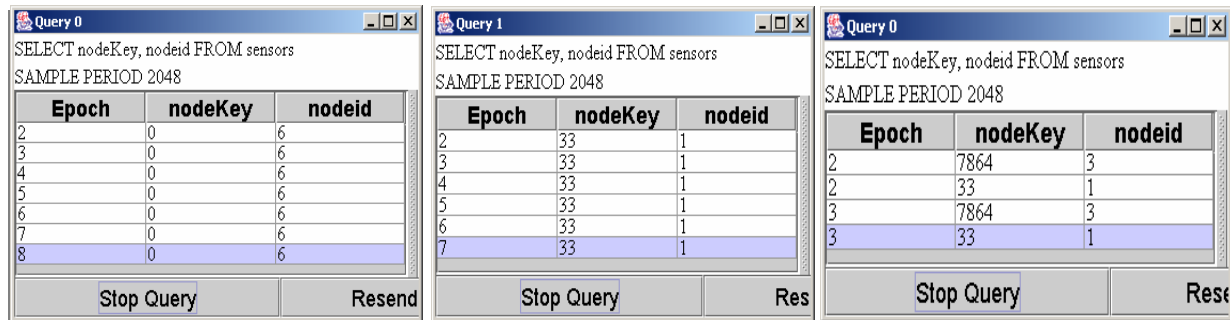


Figure 25 TinyDB+ retrieving sensor with no key (left) and one node with key (middle) and two nodes with key (right)

In the third step we compared the power utilization of querying for one key next to the power utilization of querying for any other global variable like the node ID. We ran the test by running the 'retrieve key' query in TinyDB+ in opposition to the 'retrieve key' query in TinyDB. The following table and graph shows the power level (in milli Joules) between the two systems.

Application	CPU idle	CPU active	Radio	Sensor Board	Flash	LEDs
TinyDB app (select)	742	111	1266	124	0	592
TinyDB + app (select)	756	125	1283	124	148	600

Table 2 Power utilization

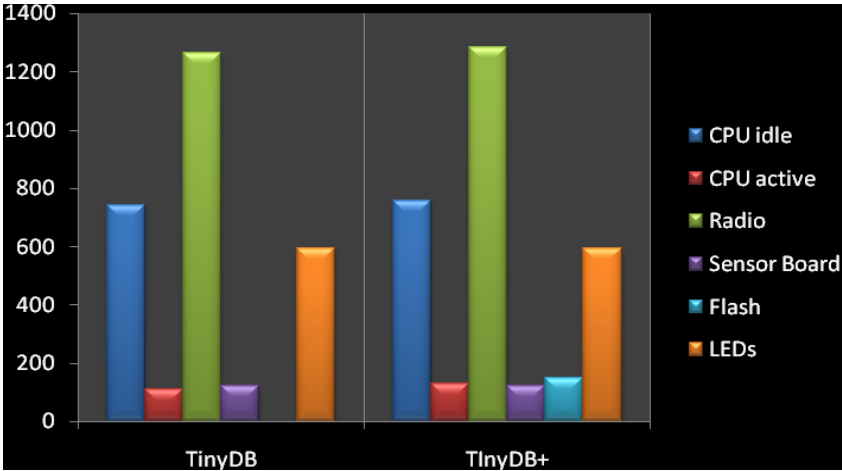


Figure 26 Power consumption differences between TinyDB and TinyDB+

The next stage was to evaluate the battery life of the new application. Normally, TinyDB lasts around 25 weeks before the voltage becomes inadequate to power the node. We found TinyDB+ to last a little bit less than that at around 23 weeks as it uses a little more power than TinyDB.

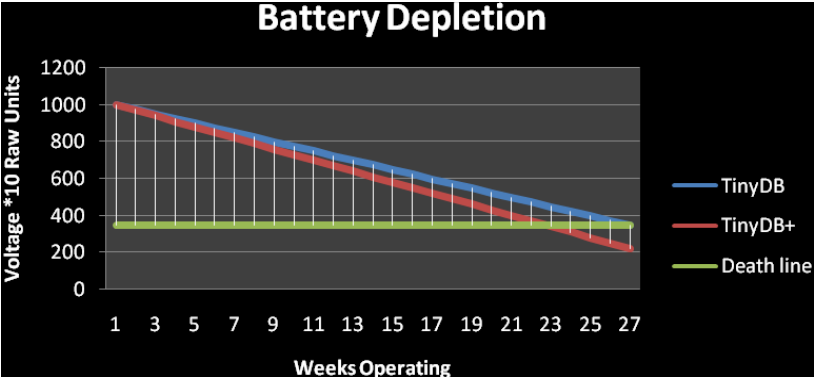


Figure 27 Comparison between TinyDB and TinyDB+ battery life

The final step was to test the functionality of changing the keys by over the air Programming. We used our army scenario to be the basis of our test. To start out we used one node which was programmed with TinyDB+. We then noted down the secret key and deployed the node. We activated the GUI and acquired the same key.

Next, we compiled a new image with a new key and activated the Deluge app. As of now we have not tied in the Deluge functionality into the TinyDB+ GUI. The node after receiving the image started to blink indicating that the rewrite process had started. After a couple of seconds the node restarted and resumed normal operation. The node before rewriting the image stored all the important information like node id and local group as well as any readings to the onboard flash. We used the GUI to issue a query to get the key again and we got what we were looking for, the new key.

Next we fabricated our own little setup by deploying a set of sensor nodes in Tossim [37] which is a full fledged simulator used to simulate wireless sensor networks. We created our own images with our fabricated key setting. Each node was loaded with three or four keys and we intentionally positioned them in a way where one section was not able to report back to the base.

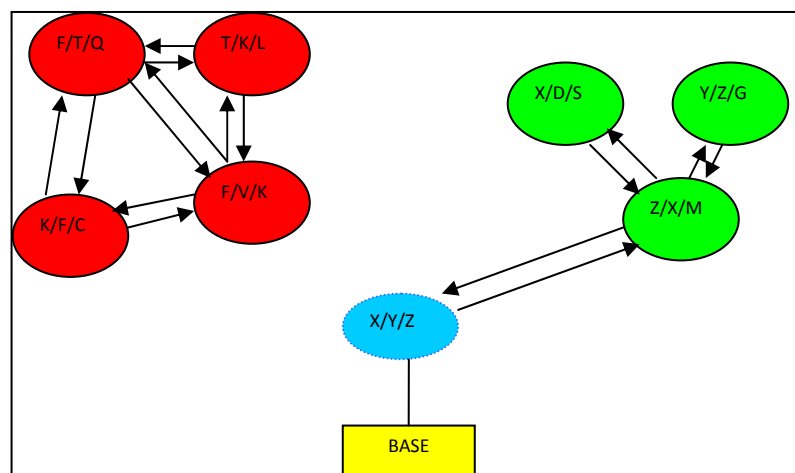


Figure 28 Fabricated deployments

From the figure we see that we started with a partially connect network. The sensors in red found common keys amongst each other and were able to establish connections. However, none of the red nodes shared a common key with the blue node which was the link to the base. All of the information went through the blue node to get back to the base station. The green nodes found a common key amongst them selves and they also found a common key with the blue node. Thus, half of the area was covered and the other half was not.

We then created an image that had a new set of keys which had something in common with the red nodes as well as the blue nodes. Our mission was to re-image the red node with the set of keys {F/V/K} with a new set of keys {F/K/X}. We then ran the Deluge command supplying the node id that we wanted to change along with the new image and waited for the process to be done.

After a couple of seconds we queried the node that received the image for its key set and received the new set. The network was fully connected and the area that was in the dark was now sending sensor reading back to base.

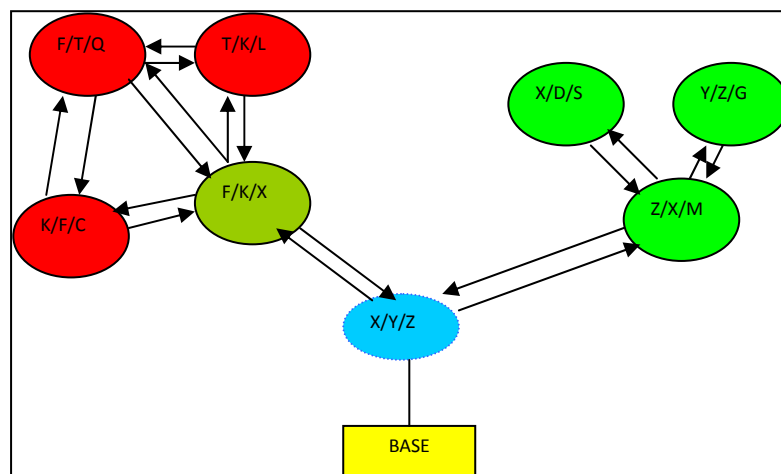


Figure 29 Topology after re-imaging

We then did a full test on all the nodes after a while to make sure there were still up and running. The results were surprising, especially when we queried for the keys each node carried. We found that they all carried the new set of keys! We first assumed that we made an error when initiating the Deluge command by not specifying which node we wanted to re-image. We modified the top level of Deluge back in chapter four to accept a node id to represent which node will accept the image. Using the history command we found that our command was correct. After many hours, we figured out that all the nodes re-imaged themselves with the new image. Even though the image was meant for only one node they all used it and rebooted. The modification to the Deluge protocol was in need of a bit more work. Looking deeper into the protocol, we found that much more modification was needed to low level side of the app. The old modification worked by sending the image to only the intended receiver. The problem was that the new re-imaged node keeps advertising what image it has. All the nodes which have TinyDB+ have Deluge wired into the application. So, the design of Deluge makes it easy to drop a new node in the network and automatically advertise its image number. If the surrounding nodes had an older version they requested the new image.

So even though the network was up and running and reporting back with sensor readings, the network from a security point of view was not safe. All the nodes had the same set of keys.

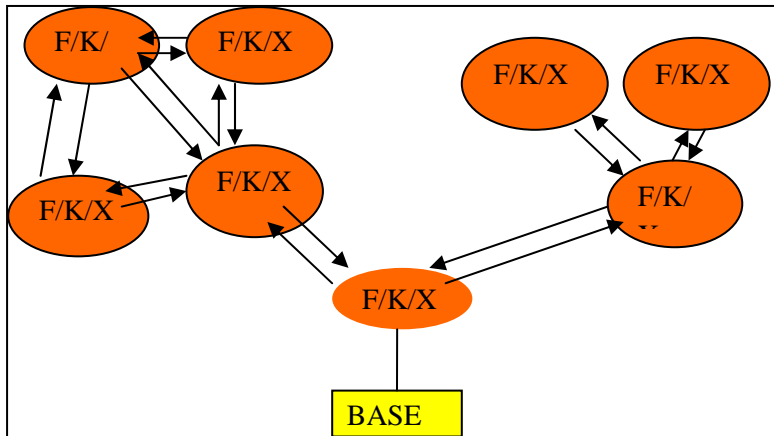


Figure 30 Topology after a couple of minutes from deploying image

4.2 Security

Our system was built on well tested and accepted methods used these days. We added the functionality of the following:

- TinySec was used to add the use of secure keys to the system. TinySec offered authentication in one mode as well as authentication and encryption in another. The security strength of TinySec can be reviewed in [31].
- Deluge was used to enable over the air programming. Over the air programming has many attack methods where an attacker can inject false images or just deplete the nodes battery by filling up its flash memory which is a very costly operation. A secure Deluge has been proposed in [38], which ensures the security issue of over the air programming. This was used in our implementation.

Chapter 5 Conclusion and future work

In conclusion, we built a system called TinyDB+ that was able to report Sensor readings and at the same time function as a key management system. TinyDB+ has the capability of retrieving the secure keys a node was given during the pre-distribution phase. We Built TinyDB+ on top of TinyDB and also enhanced the functionality of the TinyDB GUI. The system showed it still operated in the accepted threshold using the methods in [40]. Nodes with TinyDB+ installed died a bit sooner than nodes with TinyDB, the difference in life expectancy can be ignored. The system showed capability of retrieving keys or a set of keys depending on how the nodes were programmed. We showed that by using TinySec protocol authenticating and encrypting messages was possible and we showed how secure it was how ever, it is still not full proof [41]. We showed that over the air programming is possible and secure when using Secure Deluge. Deluge was modified to support only one node at a time but we found that the underlying link layer protocol needed to be fixed for it work correctly with out having all the nodes in the network reboot themselves with the image.

In the future we see our system growing into a widely accepted application that will succeed TinyDB. We will also work on adding the Deluge functionality right into the TinyDB+ main application that runs on the PC. Future work will also include automating the process where a lonely node is identified and re-imaged with a new set of keys based on the neighboring nodes' keys. We will also enable the use of TinySec commands to change the Keys and to enable the 'Secure Flash' functionality [42]. We will also add a better localization algorithm to better help display the true topology of the network.

References

- [1] Gregory J. Pottie, "Wireless sensor networks," in IEEE Information Theory Workshop Proceedings, June 1998.
- [2] G.J. Pottie and W.J. Kaiser, "Wireless integrated network sensors," Communications of the ACM, vol. 43, no. 5, pp. 51–58, May 2000.
- [3] Huang, D., Mehta, M., Medhi, D., and Harn, L. 2004. Location-aware key management scheme for wireless sensor networks. In 2nd ACM workshop on Security of Ad Hoc and Sensor Networks. pp. 29-42, May 2004.
- [4] Seyit A. C, amtepe and B" Ulent Yener Key Distribution Mechanisms for Wireless Sensor Networks: a Survey. Rensselaer Polytechnic Institute. April 2007 .
- [5] Deng, J., Han, R., and Mishra, S. 2003a. Enhancing base station security in wireless sensor networks. Tech. Rep. CU-CS-951-03, Department of Computer Science, University of Colorado. April 2003.
- [6] Chen, M., Cui, W., Wen, V., and Woo, A. 2000. Security and deployment issues in a sensor network. Ninja Project: A Scalable Internet Services Architecture, Berkeley August 2000.
- [7] Security architecture for the Internet Protocol. RFC 2401, November 1998.
- [8] Sun Dong. Mei He Bing Review of Key Management Mechanisms in Wireless Sensor Networks. November 2006.
- [9] C. Karlof, D. Wagner, "Secure routing in wireless sensor networks: attacks and countermeasures", Ad Hoc Networks, Vol.1 (2003) pp. 293 -315.
- [10] Carman, D., Matt, B., and Cirincione, G. Energy-efficient and low-latency key management for sensor networks. In 23rd Army Science Conference 2002
- [11] Chan, H., Perrig, A., and Song, D. Random key predistribution schemes for sensor networks. In IEEE Symposium on Research in Security and Privacy. April 2003
- [12] Du, W., Deng, J., Han, Y., Chen, S., and Varshney, P. 2004. A key management scheme for wireless sensor networks using deployment knowledge. In IEEE Infocom'04.
- [13] Laurent Eschenauer and Virgil D. Gligor. A key-management scheme for distributed sensor networks. In 9th ACM Conference on Computer and Communication Security (CCS), November 2002.
- [14] Blundo C, Santix A D, Herzberg A, Kутten S, Vaccaro U, Yung M. Perfectly-secure key distribution for dynamic conferences. In: Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology, Berlin: Spring-Verlag, 1992. 471_486.

- [15] Bruno Dutertre, Steven Cheung, and Joshua Levy. Lightweight key management in wireless sensor networks by leveraging initial trust. Technical Report. SRI-SDL-04-02, SRI International, April 2004.
- [16] Blom, R. 1985. An optimal class of symmetric key generation systems. In Eurocrypt 84. Blundo, C., Santis, A., Herzberg, A., Kutten, S., Vaccaro, U., and Yung, M. 1992.
- [17] Zhu, S., Xu, S., Setia, S., and Jajodia, S. 2003. Establishing pairwise keys for secure communication in ad hoc networks: a probabilistic approach. In 11th IEEE International Conference on Network Protocols (ICNP'03).
- [18] H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In IEEE Symposium on Research in Security and Privacy, 2003.
- [19] F. Xue and P. R. Kumar. The number of neighbors needed for connectivity of wireless networks. In Wireless Networks, 2004.
- [20] Crossbow technology inc. <http://www.xbow.com>.
- [21] TinyOS. <http://www.tinyos.net>.
- [22] TinyDB. <http://www.tinydb.net>.
- [23] Philip Levis. TinyOS Programming. June 28, 2006.
- [24] <http://telegraph.cs.berkeley.edu/tinydb/> .
- [25] <http://www.postgresql.org/>.
- [26] Haowen Chan, Adrian Perrig, and Dawn Song. Random key predistribution schemes for sensor networks. In IEEE Symposium on Security and Privacy, May 2003.
- [27] Wenliang Du, Jing Deng, Yunghsiang S. Han, and Pramod K. Varshney. A pair wise key pre-distribution scheme for wireless sensor networks. In 10th ACM Conference on Computer and Communications Security (CCS), October 2003.
- [28] Joengmin Hwang. Yongdae Kim. Revisiting Random Key Predistribution Schemes for Wireless Sensor Networks.2005.
- [29] Eugene Shvets. Extending TinyDB: Creating Custom Aggregates May 2005.
- [30] Wei Hong .Sam Madden. TinySchema: Managing Attributes, Commands and Events in TinyOS. Version 1.1. September, 2003.
- [31] Chris Karlof. Naveen Sastry. David wagner. TinySec: A link layer security architecture for wireless sensor networks. Pp.162-175, 2004.
- [32] Chris Karlof. Naveen Sastry. David wagner. TinySec: User Manual. 2004.
- [33] J. Jeong, S. Kim, and A. Broad. Network Reprogramming. University of California at Berkeley, Berkeley, CA, USA, August 2003.

- [34] Adam Chlipala. Jonathan Hui. Gilman Tolle. Deluge: Data Dissemination for Network Reprogramming at Scale. Fall 2003.
- [35] Kulkarni, Sandeep S.; Arumugam, Mahesh. Infuse: A TDMA Based Data Dissemination Protocol for Sensor Networks. MICHIGAN STATE UNIV, 2004.
- [36] S. S. Kulkarni and L. Wang. MNP: Multihop network reprogramming service for sensor networks MSU-CSE-04-19, Michigan State University, 2004.
- [37] Philip Levis and Nelson Lee. TOSSIM: A Simulator for TinyOS Networks. September 17, 2003.
- [38] P. K. Dutta, J.W. Hui, D. C. Chu, and D. E. Culler, "Securing the Deluge Network Programming System," in Proc. Intl. Conf. Inf. Proc. Sensor Networks (IPSN'06), Nashville, TN, Apr 2006.
- [39] William A. Arbaugh, David J. Farber and Jonathan M. Smith, "A Secure and Reliable Bootstrap Architecture," in Proceedings 1997 IEEE Symposium on Security and Privacy, pp. 65–71, May 1997.
- [40] Prasanth Ganesan, Ramnath Venugopalan, Pushkin Peddabachagari, Alexander Dean, Frank Mueller, Mihail Sichitiu .Analyzing and Modeling Encryption Overhead for Sensor Network Nodes . North Carolina State University, NC. September 19, 2003.
- [41] Carl Hartung, James Balasalle, Richard Han. Node Compromise in Sensor Networks: The Need for Secure Systems .University of Colorado, Boulder. January 2005.
- [42] Neerja Bhatnagar. Ethan L. Miller. Designing a Secure Reliable File System for Sensor Networks.October 29, 2007.

Vita

Noor Ottallah was born in Metairie, Louisiana and spent most of his life living in Israel. He then came back to Louisiana and received his B.S from the University of New Orleans. He then got married on August 12th, 2007. Noor is currently working as a level IV Programmer for e-venture technologies which is a contracting company that works for the Department of Defense.