ROBUST DYNAMIC REPROGRAMMING OF WIRELESS SENSOR NETWORKS

By

RASHMI PARTHASARATHY

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

MASTER OF SCIENCE IN COMPUTER SCIENCE

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

DECEMBER 2009

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of

RASHMI PARTHASARATHY

found it satisfactory and recommends that it be accepted.

Dr. Behrooz Shirazi (Chair)

Dr. WenZhan Song

Dr. Min Sik Kim

ACKNOWLEDGEMENTS

This thesis is the result of many collaborations, support, motivation and help from colleagues and friends.

First and foremost, I would like to express my sincere gratitude towards my advisor Dr. Behrooz Shirazi for providing me this wonderful opportunity to work under his guidance. Without his trust, invaluable support and guidance, this thesis would not have been possible. He has been a great source of inspiration and constant support. I am grateful to my co-advisor Dr. WenZhan Song for providing his support and guidance. His expertise in the system helped me achieve my goals. I would like to thank my committee member Dr. Min Sik Kim who guided me with his valuable inputs during my thesis work.

I would like to acknowledge Mr. Junaith Ahmed Shahabdeen from Intel and Mr. Razvan Musaloiu-E from Johns Hopkins University for providing help during my thesis work.

Most importantly I would like to thank my wonderful parents who stood by me and provided me with emotional and moral support. I am at this juncture today because of their constant belief in my abilities. I am also grateful to my brother who has been a role model since my childhood. Without his backing, understanding and enough patience to guide me at every step, Masters degree would not have been possible. I would like to appreciate my sister-in-law Sushma for being a good support.

I would like to mention my friend SreeNandan for all the help, laughter and surprises; and also for standing by me during good times and bad. I would like to acknowledge all my uncles, aunts and cousins for their support, encouragement and love.

I am indebted to my team mate Nina Peterson who has been a source of inspiration during this project. I would also like to thank other members of our Sensorweb research laboratory Renjie Huang, Mingsen Xu, Xiaogang and Lohith Rangappa for helping during my thesis work.

Last and definitely not the least, I would also like to acknowledge all my friends Prathibha, Ameen, Ajeet, Nancy Shah, Kulbir Singh, Sreekanth Akarapu, Rupasree, Divya and Srividhya for being my support system.

**ROBUST DYNAMIC REPROGRAMMING OF
WIRELESS SENSOR NETWORKS**

Abstract

by Rashmi Parthasarathy, M.S.
Washington State University
December 2009

Chair: Behrooz A. Shirazi

A Wireless Sensor Network (WSN) is a resource constrained network with active research in progress. WSNs deployed in remote locations are comprised of a set of devices that sense the surrounding environment and route back the data to the central repository. Some of the main challenges that WSNs face are reliability, flexibility and security. Often the network needs to be updated with error free software. When WSNs deployed in remote locations need to be reprogrammed, environmental conditions often make it impossible to physically retrieve them. Over the Air Programming (OAP) plays an important role in achieving this task. OAP is a method of distributing new software updates to devices that are not physically accessible [1]. Several platform dependent protocols have been developed for OAP. These protocols are mainly dependent on hardware architecture. As a result, any change to the hardware requires modifications to the protocol. In this work we redesign part of Deluge [2] and implement it on iMote2 platform. This work has been developed specifically for our research project called OASIS. OASIS monitors volcanic activity on Mount St. Helen. As the environmental conditions on the mountain make it difficult to access the nodes, we developed an OAP protocol to reprogram the network. We faced many challenges during the development like file system management, boot loader and Java tool chain. As the environment is harsh and rugged, it is critical for us to ensure that OAP is robust. In the context of OAP in WSNs, a system is said to be "robust" if it is capable of successfully updating itself with the latest software over the air. We evaluate performance of our implementation of Deluge on a real test bed of sensors. Our results show that Deluge successfully reprograms the nodes in the network. We also present the results of reprogramming partial network deployed on Mt. St. Helen. We further evaluate the robustness of Deluge by having different version numbers in the system and also by performing multiple binary image downloads. Our results show that Deluge is 100% robust in the lab environment. This work also contributed to developing a GUI to monitor the reprogramming of the network in OASIS.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

DEDICATION

To my parents, my brother Vijay Mandayam Vokkarne
And
My cousin Late. Naveen Mandayam Vokkarne

CHAPTER ONE

INTRODUCTION

Wireless Sensor Network (WSN) has been gaining popularity ever since it's advent as a low-cost, easily affordable means to monitor environmental conditions. Due to recent advancement in technology, WSN has been used in many different environments like habitat monitoring [3], battlefield surveillance [4], health care monitoring [5] and space exploration [6]. Most of these applications are interested in monitoring and collecting information on the behavior of the environment. Hence data aggregation is a functionality of WSN. As applications are enhanced, a need to reprogram the network becomes necessary. Reprogramming through wireless medium is in vogue.

Many protocols have been developed for dissemination of binary images [7] like Deluge [2], MOAP [8], MNP [9] and so on. The above protocols function well on certain hardware. Once the hardware changes, these protocols cease to function. The hardware change could be anything from new radio being used for communication, flash memory change to changing the entire microprocessor. The slightest modification in the hardware requires the protocol to be modified. Our focus is to develop an OAP protocol that works on the latest hardware available called iMote2. The aim of this thesis is to develop an OAP protocol to reprogram WSN for iMote2 platform and also ensure that such a protocol is robust.

1.1. OAP

In general, OAP can be defined as the method of distributing new software updates. The OAP mechanism requires the existing software and hardware of the device to support features to receive and install new software via the wireless medium.

Due to erroneous implementation or requirement of new features, reprogramming nodes becomes imperative. Advances in the field of technology have enabled to provide software patches to the applications through reprogramming. WSN deployed in remote areas often make it impossible to physically retrieve them. Also, manually reprogramming each node by plugging in the device to the computer or a PDA is not feasible for a large network. Therefore, a need to reprogram a network of nodes through the wireless medium becomes a necessity.

1.2. Robustness

With the rapid growth of wireless communications, wireless networks are deployed widely. However, due to the characteristics of wireless networks such as high error rate, limited bandwidth, time-varying channel conditions, limited battery of wireless devices and the diversity of wireless networks and devices; wireless reprogramming of the network becomes challenging. A deployed network of nodes is expected to have many qualities like robustness, stability, efficiency and so on throughout their lifetime. Robustness is the quality of being able to continue to operate despite abnormalities in input, calculations etc.

1.3. Motivation

Monitoring environmental conditions requires a real-time information gathering system. The system also requires being situation-aware and should operate unattended for long periods of time. As our project OASIS [10] is a system that is deployed to monitor mount St. Helen, a simple and easy protocol to reprogram the network over the air is required.

1.3.1. Optimized Autonomous Space In-Situ Sensor-web (OASIS)

OASIS [10] is a volcanic hazard monitoring In-Situ sensor-web comprised of smart sensors capable of making autonomous data acquisition decisions. OASIS was developed by teams [11] from U.S. Geological Society (USGS), Jet Propulsion Laboratory (JPL) and Washington State University (WSU). OASIS is a dynamic, scalable sensor-web with two-way communication capability as depicted in Figure 1. The sensor-web autonomously determines the topology and node bandwidth (stage 1). Upon seeing the activity level increase, the network topology self-organizes and a request to re-task the space assets is sent (stage 2). When the sensed data is acquired, it is fed back to the control center (stage 3). Finally the in-situ sensor-web ingests remote sensing data and re-organizes accordingly (stage 4).

The system consists of a ground component and a space component. The ground component comprises of a network of nodes that communicate with each other, collect data and route the data to the central repository. Figure 2 provides a high level design of the network deployed on mount St. Helen.

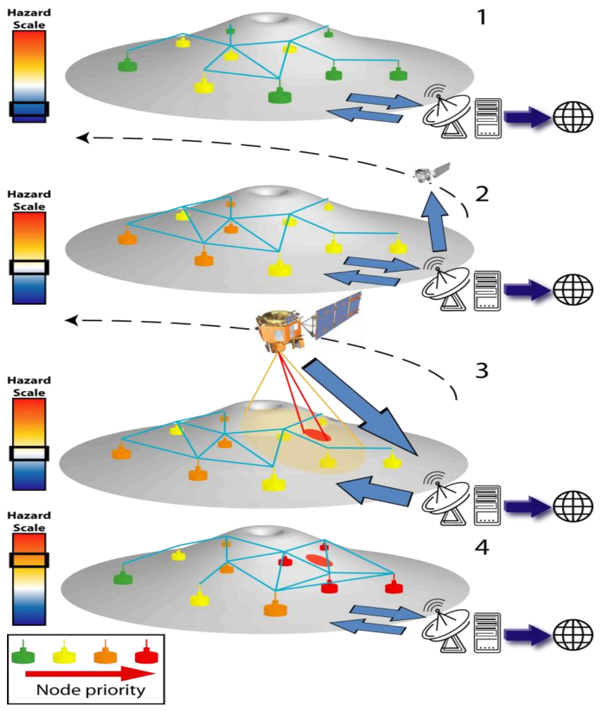


Figure 1 Optimized Autonomous Space - In-situ Sensor-web concept

The sensor nodes deployed form logical clusters for network management and situation awareness. The data flow forms a dynamic data diffusion tree rooted at the gateway. The network adjusts the bandwidth and power smartly according to environmental changes and mission needs. Remote control center manages network and data, and also interacts with space asset EO-1 and Internet providing a feedback loop to the network.

Our WSN is comprised of iMote2 sensors attached with a seismic sensor, an infrasonic sensor and a lightening sensor. The network will be deployed for one year. We had to overcome many difficulties to deploy the network for this length of time like the nodes had to withstand the harsh environmental conditions and the sensors had to be powered for a whole year. USGS developed *“Spider”* (as shown in Figure 3) which can all the devices along with the car battery required for a whole year.

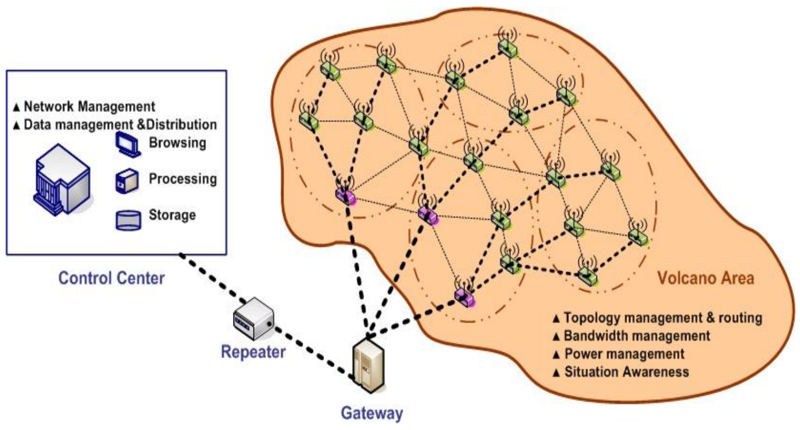


Figure 2 In-Situ Sensor Web architecture

These *spiders* can withstand harsh environmental conditions and can be deployed with the help of a helicopter. For the network to be robust and autonomous many components were developed by the students at Washington State University. Some of the major contributions are design of context and situation awareness [12], development of tremor detection and MAC protocol for efficient routing [13]. OAP was another major accomplishment in this project. The details of OAP will be discussed in the remainder of this work.

1.3.2. OAP

Protocols developed so far for OAP like Deluge [2], MNP [9] and MOAP [8] are all functional on Tmote Sky [14] or MicaZ [15] mote platform. Allen et. al. [16] deployed a network of Tmote Sky sensors with OAP enabled on Reventador mountain at Ecuador for a 19-day period. They also present results related to OAP in this paper. As most of the OAP protocols developed work on Tmote Sky and MicaZ platforms, upgrading the hardware from Tmote Sky to iMote2 made these protocols non-functional.

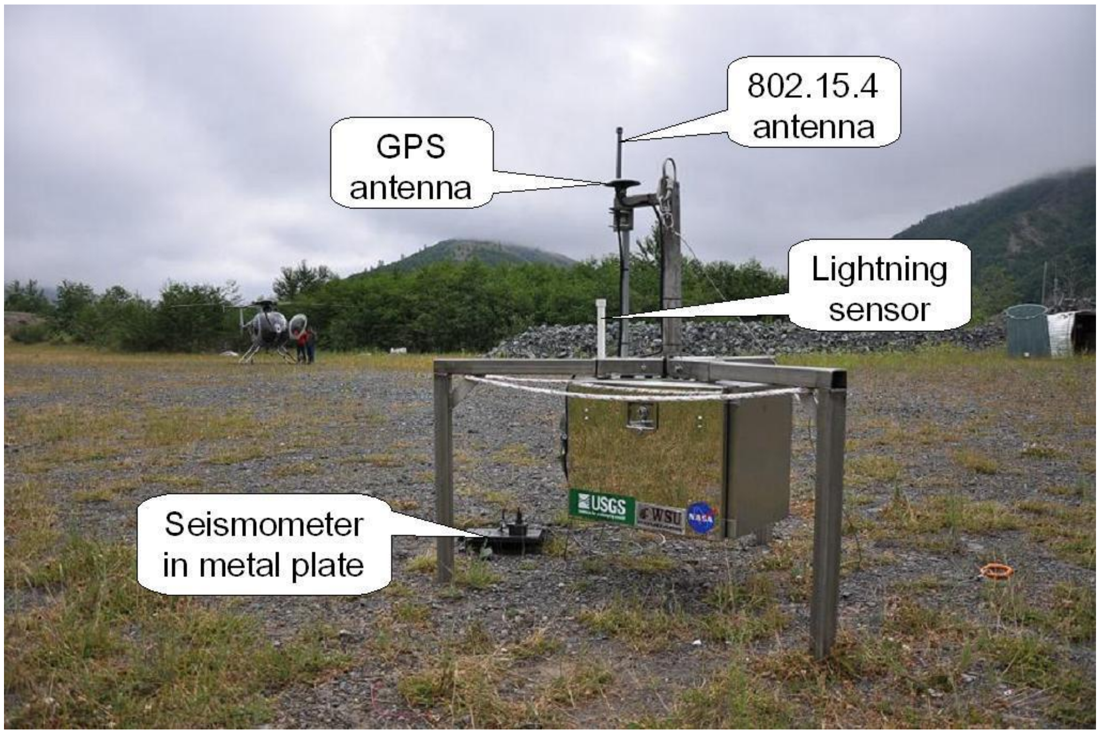


Figure 3 Spider

A generic wireless reprogramming architecture is shown in the Figure 4 [7]. All the reprogramming protocols should have these basic functions incorporated. A version control database should maintain the version number of the program. The scope selection should allow the administrator to select the nodes for reprogramming. Data packets should be created before disseminating through the radio. On the receiver side, these packets should be decoded and validated before being written to the internal or external flash. A protocol to transmit and receive the data packets in the form of updates from source node to intended target nodes should be developed.

Before it can be executed, the new program should be received in its entirety without errors. The verification should be done in the validation segment. If a node needs to acquire new code, it should send code acquisition requests. Trigger conditions can be set based on the requirement. A node should switch to the new program typically through a reboot command.

A boot loader, which is located in a specific region of a microcontroller, should copy the new program from flash memory segment to the program memory section. Before loading the new program, the boot loader should do some bookkeeping and preserve data and states. The management interface should provide options to monitor start or cancel the data dissemination.

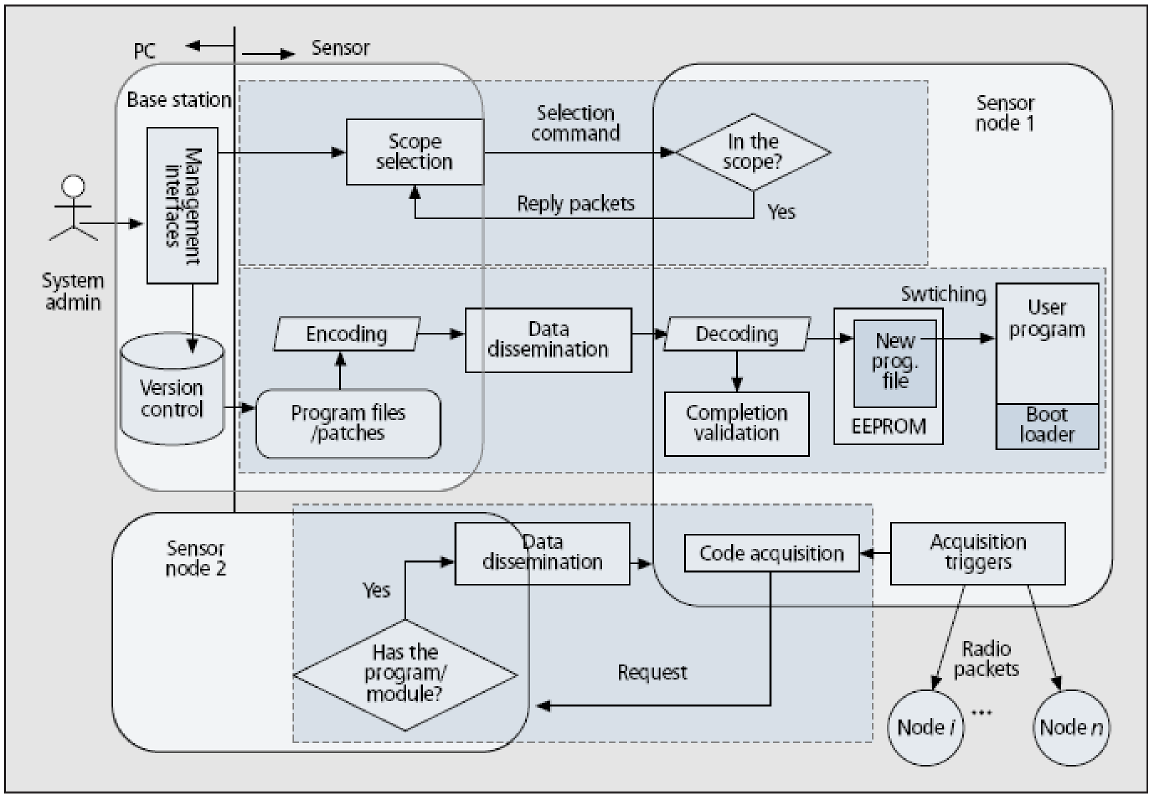


Figure 4 WSN Reprogramming Framework[7]

All OAP protocols follow the above architecture to obtain the required functionality. In this novel work, we had to implement one such protocol for iMote2 platform. More discussion regarding the different protocols is provided in Chapter 2.

1.3.3. Robust OAP

OAP is one of the fundamental services that rely on broadcast mechanism. Existing OAP protocols are unreliable with respect to communication, communication delay, and energy in harsh, unpredictable broadcast environments with software/firmware constraints. All the protocols developed for code dissemination are epidemic in nature. They use broadcast mode of communication. These protocols do not restrict the origin of a code update. Andrew Hagedorn et al [17] attempt to overcome the drawback of OAP and provide a robust OAP mechanism. An OAP is said to be robust if it can endure the various abnormalities in the functionality of a stable network and yet reprogram all the nodes in the network. We define robustness for our protocol based on the number of nodes that are successfully reprogrammed in the network.

In this novel work we attempt to make Deluge robust to firmware and software failures. More information is provided in chapter 3.

1.3.4. Deluge GUI Tool

Nodes in our OASIS network collect various types of data and route it to the gateway node which in turn forwards it to the control center. The control center consists of a set of latest server computers that are dedicated to collect the data for postmortem analysis. Geoscientist and space scientists use a diverse range of software tools on this data set to comprehend the behavior of the volcano and to control the sensor-web. Also there are many graphical user interface (GUI) developed for OAP on wireless terminals. We developed one such software tool to provide visualization for OAP. This standalone application allows tracking the progress of the dissemination on the gateway node and displays the list of applications already present on the flash memory of the node. We will be discussing it in more detail in the following chapters.

This chapter provided a brief description of how OAP has emerged as a solution to network reprogramming. It also highlighted how this protocol aims at providing an easy reprogramming method but suffers from a major drawback with the hardware being the single point of failure. It also further discussed the need for an OAP protocol for iMote2 platform. Chapter 2 describes some of the existing protocols for achieving network reprogramming. It also provides a detailed description of one of the protocol, Deluge, which is the aim of this thesis. Chapter 3 provides a detailed description of the proposed Deluge protocol design and its modifications. Chapter 4 discusses the performance evaluation of Deluge on a real test bed of iMote2 sensors and also evaluates the robustness of the system. Chapter 5 concludes by observing the merits and demerits of the protocol. Finally Chapter 6 elaborates the future scope on this work.

CHAPTER TWO

BACKGROUND AND RELATED RESEARCH

2.1 Network Reprogramming Protocols

Several protocols have been developed for dynamically reprogramming WSN. XNP [18] is the first network reprogramming protocol developed by Crossbow and it works in two phases. Firstly, the program is downloaded and stored on the flash memory. Secondly, In-System programming and rebooting the new application is performed. During the download phase, the node handles different commands and downloads the program to the external flash. In-System processing allows the boot loader to sequentially read the image from a user-defined location in the external flash and program the main memory. It further reboots the processor to start executing the new application. The main disadvantage of this protocol is that the reprogramming can be done if the nodes are one-hop away. As most networks are multi-hop, this protocol becomes inefficient.

MOAP [8] unlike XNP deals with dissemination of code over a multi-hop network. It is a comprehensive approach to network reprogramming. Single hop dissemination is recursively extended for Multihop. The code is disseminated hop by hop. MOAP uses NACK based transmission, broadcast and unicast mechanisms. It further uses a sliding window to manage the segments that are required for efficient retransmission. The main disadvantage of MOAP is that if a node does not receive the entire image it cannot disseminate further.

In Trickle [19], the metadata of the images are periodically transmitted by a node if it has not heard from any other nodes in its vicinity. Metadata sometimes called meta-information is the information regarding the binary image. The metadata transmitted periodically leads to either an update of the code or the knowledge that all its neighbor nodes are up to date. Trickle maintains a constant communication rate at all times and transmits at most once every second to avoid collisions and packet loss. The disadvantage of Trickle is that it addresses only single packet dissemination.

MNP [9] uses the sender selection protocol where the next node to transmit in a neighborhood is decided by the number of download requests received at each node. This protocol keeps the number of senders in a neighborhood to minimum, thereby reducing the energy consumption. A sender selection algorithm selects a node in the neighborhood that can have maximum impact. As a result, MNP avoids the hidden terminal problem faced by other protocols. A bitmap is maintained at the receiver end to detect loss of packets. The advantage is that the radio does not have to be on all the time. When a node becomes a sender, all the other potential sender nodes will transition into sleep mode for a predetermined time. The main drawback of MNP is that it incurs additional communication overhead to choose the sender.

The FireCracker [20] disseminates small pieces of data rapidly in the network. It combines broadcast and routing in combination to deliver packets rapidly in the network. Firecracker routes the packets to several distant destination nodes. Once the data has arrived at the destination, it simultaneously starts disseminating the packets in the network by broadcasting.

Sprinkler [21] achieves reliable data dissemination by embedding a virtual grid over the network and locally computing a connected dominating set of nodes, there by avoiding redundant transmissions and reducing the number of senders. The advantage of the Sprinkler protocol is that if the number of nodes in the network increases without increase in the number of hops, it results in a steady transmission rate. The disadvantage of Sprinkler is that it is assumes that the location of the nodes are known prior to dissemination which is generally not the case.

Aqueduct [22] is a code dissemination protocol for heterogeneous networks. Aqueduct achieves robust and efficient dissemination of the code by limiting the number of nodes involved in the dissemination process. The nodes in the network are classified as either forwarding nodes (which are not interested in the code being disseminated and act merely as a node in a transmission path) or member nodes (who act as receivers requiring the new image being disseminated). The member nodes cache the entire code but the forwarding nodes may cache only a part of the code being disseminated.

Deluge [2] is a reliable data dissemination protocol that is widely used in the TinyOS environment. Deluge as the name suggests is used to inundate the network with a new image. Deluge uses many concepts of MOAP in addition to dividing the data into pages and packets. It also deals with asymmetric links. Deluge uses a three-phase handshaking protocol (advertise – request – data). Deluge periodically advertises metadata in its neighborhood so that all the nodes in the communication range can maintain the newest version of the program image available. If any node in the neighborhood does not have the latest version, a download request is sent and subsequently downloaded. Deluge divides every program image into sizeable pages and each page is further divided into packets. The major advantage of Deluge is that it dynamically adjusts its advertisement rate depending on the messages it has received. The disadvantage of Deluge is that the radio needs to be on at all times so that the nodes can listen to all the messages.

Deluge protocol is a reliable mechanism to flood the network with software updates. Deluge epidemically propagates the software code in a homogeneous network. Figure 5, Figure 6 and Figure 7 shows the different stages of Deluge. Figure 8depicts the flow chart of the protocol. The nodes periodically advertise the metadata of all the program images (Figure 5). Deluge is a receiver-initiated code dissemination protocol, thus a node responds with code updates once it receives a request to download from another node (Figure 6 and Figure 7).

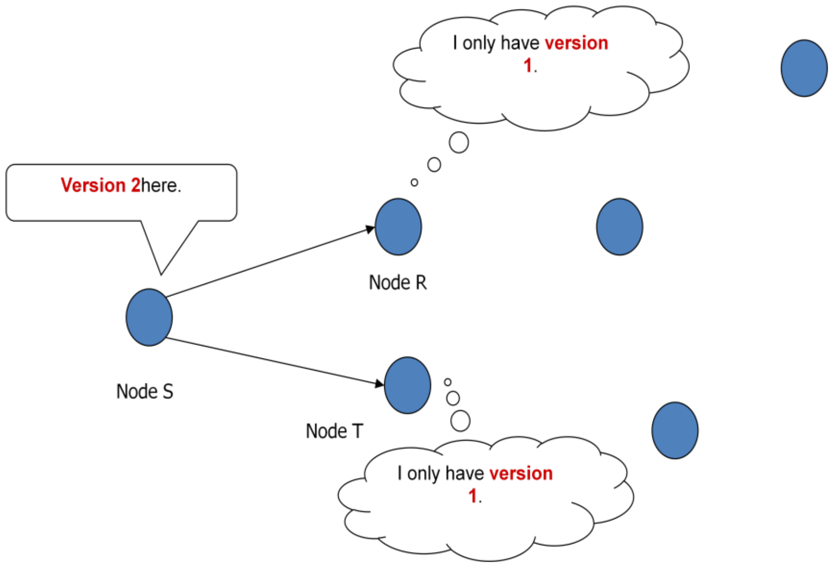


Figure 5 Advertisement Message[2]

Deluge divides large binary images to small manageable pages and packets. During the download phase, the receiver node will request page *i+1* only if complete copies of all *i* previous pages have been received. Unlike MOAP, the nodes do not need to download the entire code before disseminating further.

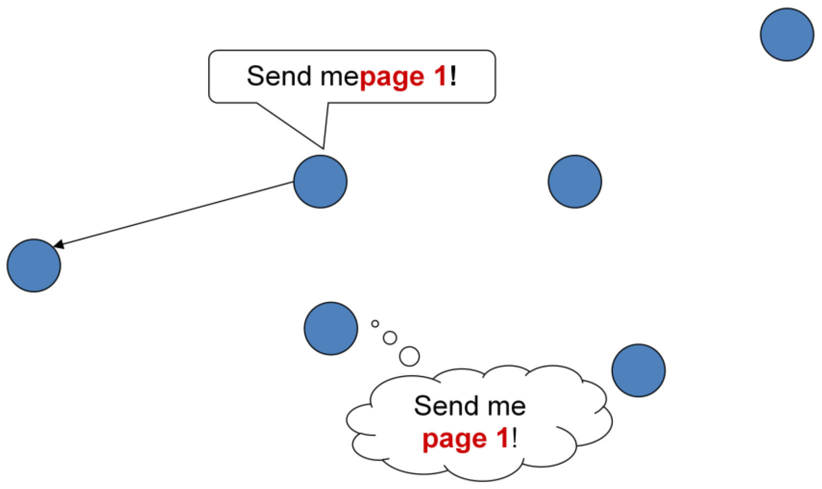


Figure 6 Request Message[2]

Once a receiver node receives a few pages, it can start disseminating it further in the network. This allows for pipelining of the download to take place. As a result, Deluge is easily scalable in dense networks.

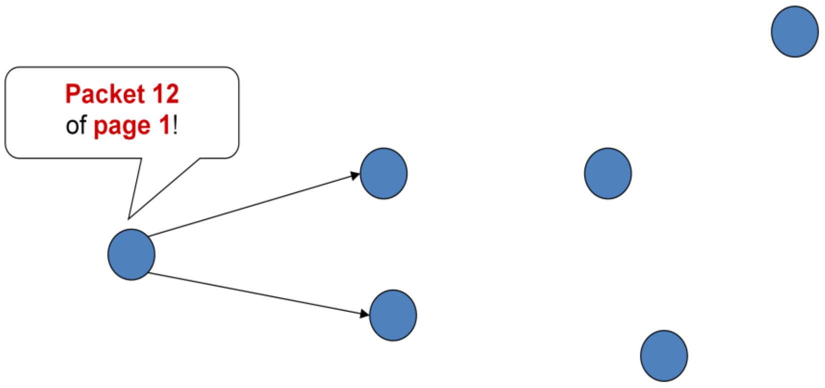


Figure 7 Data Message[2]

Deluge maintains a state within each node that allows it to recognize that its code is obsolete. Once this occurs, it will then send a unicast message to a particular neighbor node asking it to send the appropriate updates. Once the download is complete, the nodes reboot from the new image either automatically or through an explicit reboot command. Deluge allocates three files to download new code updates and can reboot from any of them.

As it is the most commonly available OAP in TinyOS environment and because of its major advantages like multi-hop support, dynamic dissemination, we chose Deluge. More details about Deluge will be discussed in Chapter 3.

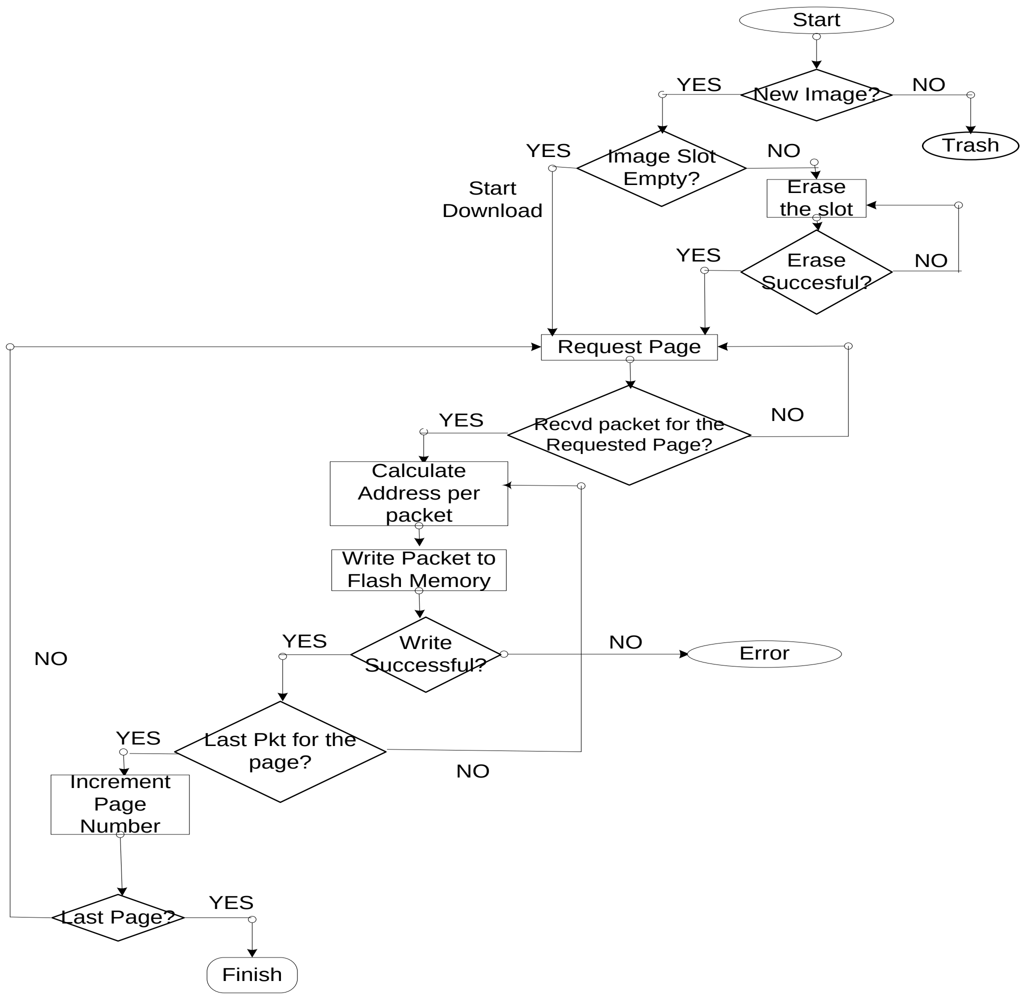


Figure 8 Flow chart Deluge Protocol

2.2 Established Robustness mechanisms in WSN

Most protocols developed for code dissemination emphasize on robustness to an extent. Some protocols insist on providing robustness through hardware and some through software. We briefly discuss some of the protocols that address robustness as an important aspect of OAP.

Lane A. Philips proposed Aqueduct [22], which provides robust code propagation for a heterogeneous network by limiting the number of nodes involved in the propagation. It also adapts robustly to changing RF quality. As aqueduct is developed based on Deluge, most of the robustness qualities from Deluge like using pull-based data dissemination, passive suppression, and soft state and adaptive response are retained. In addition to these robustness features, Aqueduct ensures robust code dissemination by creating aqueducts among the nodes participating in the download. Further Aqueduct is robust to spatially irregular, time-varying asymmetric link quality. Symmetric shortest path links are chosen to build Aqueducts instead of choosing paths that might be shorter but over unreliable links.

Robbert Van Renesse et al [23] achieve robustness through randomized peer-to-peer protocol for a client server system. Most systems developed on centralized servers are vulnerable to attacks and failures. Hence Astrolable uses peer-to-peer system by running a process on each node. These processes communicate through an epidemic protocol. Therefore, these nodes become tolerant or robust to failures. The nodes communicate using randomized point-to-point messages.

Steven D. Gribble, Matt Welsh et al [24] propose a robust architecture for Internet-scale systems and services. They propose a distributed service architecture, which is easy to construct and provides robust services. It also deals with the huge throughputs demands and availability requirements.

As robustness is defined as the ability of a network of nodes to reprogram itself successfully, we chose to consider the same parameter for our protocol.

A detailed description of the system design, implementation of Deluge and its robustness features are presented in the next Chapter.

CHAPTER THREE

SYSTEM DESIGN AND IMPLEMENTATION

3.1 System Design

TinyOS [25] is a free and open-source component based operating system designed for wireless embedded sensor networks. It is written in nesC [26], a component based, event-driven programming language. nesC is built as an extension to the C programming language with components wired together to run an application on TinyOS. Our work uses TinyOS and nesC as it is robust for software development on embedded systems. The ability of the developer to choose different components based on their requirements makes this operating system and programming language robust. Many research groups have contributed to the TinyOS library by developing different standard network protocols, drivers and so on. Our implementation contributed to the TinyOS library.

3.2Hardware

We are using the state-of-the art iMote2 devices in our project. These devices are capable of fulfilling our system requirements of low power consumption, high computing power, data precision and memory requirement. This section discusses some of the hardware used in our project and specifically for OAP.

3.2.1 iMote2

Tmote Sky, MicaZ and iMote2 are the most widely available sensor node platforms today. The Tmote Sky is built around an MSP430 microcontroller. MSP430 family of processors has an 8 MHz CPU, a 10 KB RAM and 48KB flash memory. MicaZ has a 7.37 MHz CPU with a 4 KB RAM space available.

The iMote2 [27] is an advanced wireless sensor node platform. It is built around the low powered PXA271 XScale processor which has a ChipCon CC2420, 802.15.4 radio and a built-in antenna integrated [28]. The PXA271 has the lowest voltage set at 0.85V. It can operate between 14 MHz and 104 MHz. By using dynamic voltage scaling, the operating frequency can be scaled to about 416 MHz. The PXA271 package comprises of three chips, a 32 MB SDRAM chip, a 32MB flash chip and the processor chip. Additionally, the 256KB SRAM is divided equally into 4 banks. The PXA271 also provides a diverse range of I/O options like camera interface, Infrared ports, I2C, I2S, three synchronous serial ports, 3 high speed UARTs, GPIOs, SDIO, AC97 and a USB client and host. To accelerate the multimedia applications PXA271's MMX coprocessor can be used if required. An iMote2 is as shown below in Figure 9.

Figure 9 iMote2

Due to memory constraints on Tmote Sky and MicaZ, we use iMote2 in our OASIS project.

3.2.2 Debug Board

The debug board is designed to provide dual USB serial ports and a JTAG interface for iMote2 [29]. This device can be directly plugged to the iMote2 with the advanced connector set provided. It also facilitates debugging by providing a second set of pass through connectors. This interface board can be connected to a USB host like a PC through a mini-B USB connector. The standard baud rates from 110 to 921,600 bits per second are supported. The interface board also provides a 20-pin connector that allows us to use the Intel XScale software development tool suite or similar tools for

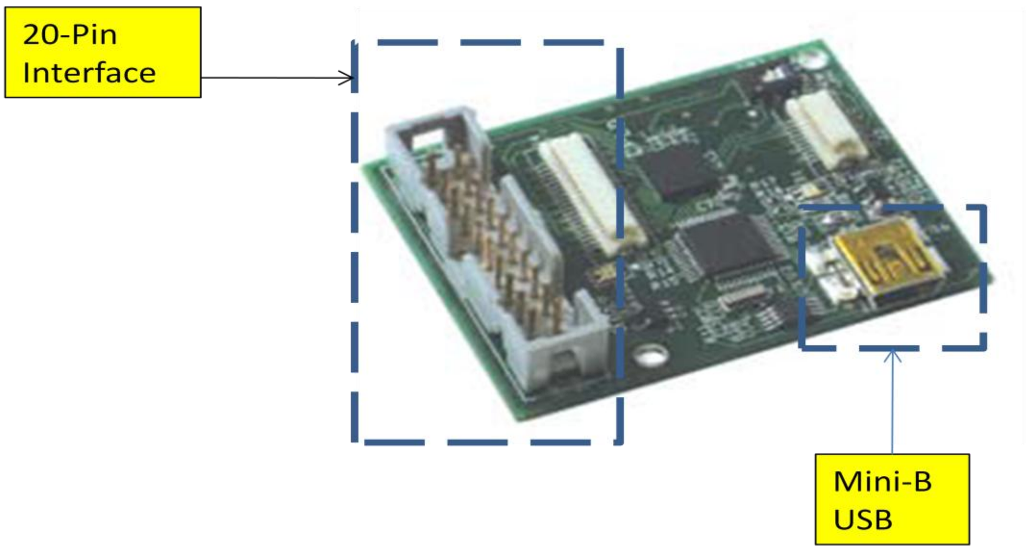


Figure 10 IIB2400 Debug Board

Software development, debugging and reprogramming the iMote2's flash memory. This board acts as an interface between the iMote2 and the PC to download the boot loader.

3.3Firmware

We used TinyOS operating system for our software development. Device drivers and other firmware required for iMote2 are available as part of TinyOS.

3.3.1Deluge for iMote2 sensor platform

Our goal was to implement an OAP protocol for iMote2 platform that takes into consideration all the hardware changes, specifically flash memory management and the boot loader. It was a big challenge to develop this protocol on iMote2. Our solution considers some of the major constraints in embedded systems like memory and power constraint. We will describe the different functionalities we implemented to make Deluge compatible with iMote2 platform [30].

As explained in Section 3.1.1, due to the increase in size of both the RAM and the flash memory of iMote2 as compared to Tmote Sky, we had to develop a sophisticated mechanism to access them. In addition, the boot loader of the iMote2 is much more sophisticated than that of the Tmote Sky.

The flash memory architecture of iMote2 is as shown in Figure 11Figure 11. The flash memory is divided into banks. Each bank consists of blocks where each block is 128 KB. Certain sections of the flash memory are intended to perform certain functionalities. We explain the functions of each section in detail before describing the implementation.

The boot loader is always part of the flash memory and is located in a protected region, which cannot be overwritten by any user defined applications. The role of the boot loader is not limited to downloading applications through the USB to the primary and the secondary image locations. In addition, the boot loader is also responsible for copying the image to the boot location after verifying the integrity of the image and executing it.

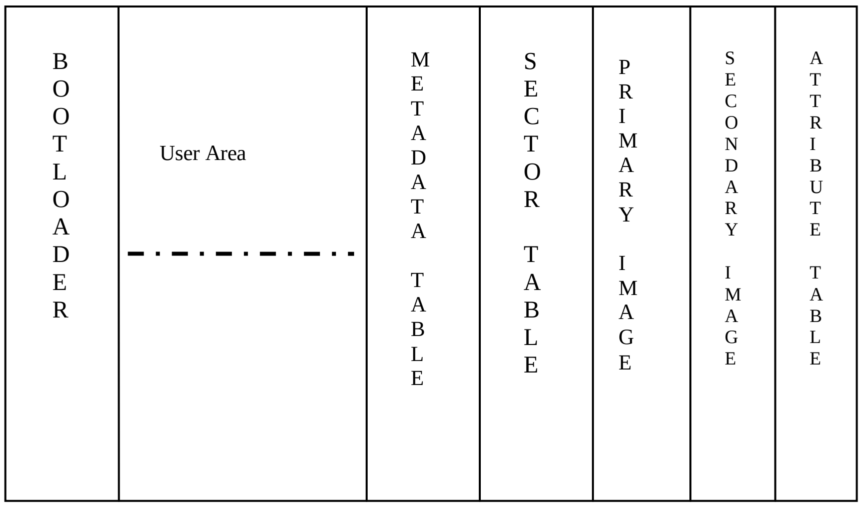


Figure 11 Default Flash Memory Map

Following the boot loader is the user area, where any user-defined application can write data. The number of flash memory blocks available for a user is approximately 200.

The file system of iMote2device is a Linux file system where each file that is created will have an associated read and write pointer with additional information. This information is stored in the Metadata table located immediately after the user area of the flash memory. Every write to a file in the flash memory results in an entry in the metadata table with the updated write pointer. The previous entry is invalidated.

As in a Linux file system, a user can create files in the user area of the flash memory. Similar to Linux file system, an entry of the files created is stored in the sector table. The sector table also holds the information corresponding to files such as the number of blocks allocated to each file, the start block number, the end block number, the total size of file and the name of the file. This table is most critical for applications that depend on files since the table is vulnerable to corruption. Every file created should have at least 128 KB space.

Applications can be downloaded to iMote2 using the Mini-B USB port. These downloaded applications are stored in the primary and secondary image locations on the flash memory. The boot loader, by default, uses these locations to load the application to the main memory. During the download process, the primary and the secondary image locations are exchanged. For example, if application A (app A) is the first application downloaded, then the primary image location will be used to download app A. Next, if app B is downloaded, the secondary image location becomes the primary image and app B is then downloaded. The location containing app A becomes the secondary location. This way, the two slots are interchanged at every download.

The attribute table is the table stored at the high end of the flash memory. The attribute table is a collection of tables that hold the different parameters used by the boot loader such as the primary and secondary Image location addresses, the size of the images in these locations, additional flags, and the boot location address. This table is a read only to the users and read/write for the boot loader. However, there is also a shared attribute table that is a portion of the attribute table that has read/write permissions for both the user and the boot loader. This table has some miscellaneous information. For example, the Cyclic Redundancy Check (CRC), which is responsible for checking the image to make sure that none of the bits, is corrupt. Additionally, the table also contains information related to reboot locations and self test information.

We now describe the implementation details for Deluge on iMote2 platform. To write any binary application onto the iMote2 flash memory, we first have to erase the user area. Since the iMote2 sensors do not contain sectors - instead they contain blocks - it was necessary to erase all 200 blocks before downloading the binary image. Because each block is 128 KB, we set the minimum size of a file created to be at least 128KB. In our case, we set the size of the file to be 1MB (the maximum size of the boot location). The file FormatStorageM.nc allocates space for individual files and updates the metadata table and the sector table with the entry of the files.

Three files are created in the user area to be used for storing the code updates of the flash by specifying the filename, size and volume id. After creating the required files, the basic functionality of Deluge should be modified. We modified the interface that interacts with the flash storage; specifically for reading and writing the binary data and verifying the cyclic redundancy check of the packet being written.

The binary application cannot directly be transmitted over the air using the Java application running on PC. Hence we modified the compiler for iMote2 applications to generate a file called *tos_image.xml* that can be easily read by the Java application on PC and sent to the base node. This file is generated using a Perl script. The *tos_image.xml* is as shown in the Figure 12. This *tos_image.xml* contains the binary data of the application and the metadata (application information).We incorporated the Perl script in the imote2 platform compilation file as shown in Figure 13.

```
<tos_image>

<ident>

<hostname>grad-lap-01</hostname>

<program_name>DelugeBasic</program_name>

<user_hash>DC9BD7ABL</user_hash>

<unix_time>4AD3A6A4L</unix_time>

<uid_hash>1ADF1793L</uid_hash>

<user_id>npicone</user_id>
```

```
<platform>imote2</platform>

</ident>

<image format="ihex">

:10000000070000EA0B6100EA0B6100EA812C00EABC

….

….

….

</image>

</tos_image>
```

Figure 12tos_image.xml

The binary image of the iMote2 is much larger than that of the Tmote Sky. For instance, a simple program like Blink (available in TinyOS) that periodically toggles the LED is about 10 pages in Tmote Sky where as it is about 28 pages for iMote2 platform. As a result the download time for each binary image had to be decreased significantly by increasing the number of bytes per packet. We increased the number of bytes per packet from 23 bytes to 50 bytes.

```
MAIN_IHEX = $(BUILDDIR)/main.ihex
….
ifndef BUILD_DEPS
ifeq ($(filter $(BUILDLESS_DEPS),$(GOALS)),)
   BUILD_DEPS = bin tosimage bytes $(POST_BUILD_EXTRA_DEPS)
endif
endif
setid: FORCE
	@cmd () { echo "$$@"; $$@; }; if [ x = x$(NODEID) ]; then cmd
$(OBJCOPY) --output-target=binary $(MAIN_EXE) $(INSTALL_BIN);
else cmd $(SET_ID) --objcopy $(OBJCOPY) --objdump $(OBJDUMP) --
target binary $(MAIN_EXE) $(INSTALL_BIN) $(NODEID); fi

tosimage: ihexbuild_tosimage FORCE
	@:
ihex: exe FORCE
	$(OBJCOPY) --output-target=ihex $(MAIN_EXE)
```

Figure 13 iMote2 Compilation Rules

Every write to any file in the flash memory results in an entry in the metadata table updating the current write pointer. The size of the metadata table in the original design (of the iMote2 sensor) was two blocks, equivalent to 256KB. When a reboot occurs, the entry in one block is copied to the next block as well as the RAM. The first block is then erased. This process repeats upon every reboot. The algorithm is as shown in Figure 14.

```
Initialize_storage
i = metadata_table_start_address
for i < metadata_table_end_address
if  IS_VALID_METADATA_BLOCK(i) = true
break;
end if
end for
k = i + FLASH_BLOCK_SIZE //Size of one block
while i< k
for j < number of files in sector table
m = read (size of metadata from i)
if IS_VALID(m)
      Copy data to RAM (linked list)
       Increment j
else
i += sizeof(metadata)
end if
if j = number of files in sector table
if(files found in block starting with address i)
erase block i
insert copied data to i+1
else
```

```
erase block i+1
insert copied data to i
return
end if
else
return FAIL
end if
end for j
end while
end initialize_storage
```

Figure 14 Pseudo code for Metdata table access

As the binary image size is large and a significant number of metadata entries are generated, the size of the metadata table was not sufficient. Therefore, we increased the size of the table to be large enough to accommodate the entries. We further disabled the feature of switching between the two metadata blocks. We now directly read from all the blocks and update the entry in the RAM accordingly. This is described in Figure 15.

To boot the binary image downloaded over the air, we modified the boot loader. The boot loader always loads the image to main memory from the pre-defined primary and secondary locations. The boot loader also checks the integrity of the image before it is copied to the bootable location. In addition to that, the boot loader requires additional information such as size of the image and address of the image. The Tmote Sky architecture contains a memory segment called Information memory (also called internal flash) of size 256 bytes, which is divided into half to allow byte-level, read and write operations. This information memory is required to pass

```
initialize_storage
i = metadata_table_start_address
for i<metadata_table_end_address
//The size of this table has been increased from 2 blocks to 5
for j < number of files in sector table
m = read (size of metadata from i)
if IS_VALID(m)
        Copy data to RAM (linked list)
        Increment j
else
i += sizeof(metadata)
end if
if j = number of files in sector table
erase all blocks from metadata_table_start_address to
metadata_table_end_address
insert copied data at metadata_table_start_address
return
else
return FAIL
end if
end for j
end for i
end initialize_storage
```

Figure 15 Modified pseudo code for Metadata table access

information to the boot loader. However, iMote2 does not possess an internal flash. As a result, we developed a component called Netattr (Network Attribute) to pass the required information to the boot loader. This component runs with Deluge and writes data such as address and size of the image to the shared attribute table and verifies the CRC of the image before signaling a reboot. The pseudo code for the functionality of Netattr is provided in Figure 16.

```
Signal_Reboot
Verify the address location passed (x)
if valid
 Verify the integrity of the entire image x
iffound_valid
    Read the shared table to RAM
    Update the shared table with new information about the image
    Erase shared table and write the new shared table back to the flash
memory
Call reboot
  End if (found_valid)
End if (valid)
 End Signal_Reboot
```

Figure 16Netattr Functionality

Once the boot loader has all the required information, we developed an approach to load the image from user-defined location to the main memory in such a way that the default method of loading the application from primary and secondary location is held intact. We designed it as shown in the Figure 17.

The boot loader also checks the integrity of the image before it is copied to the bootable location. This CRC check requires significant RAM space, thus we use the CRC check (that is part of Tmote Sky's Deluge) and disable the CRC check done in the iMote2 sensor.

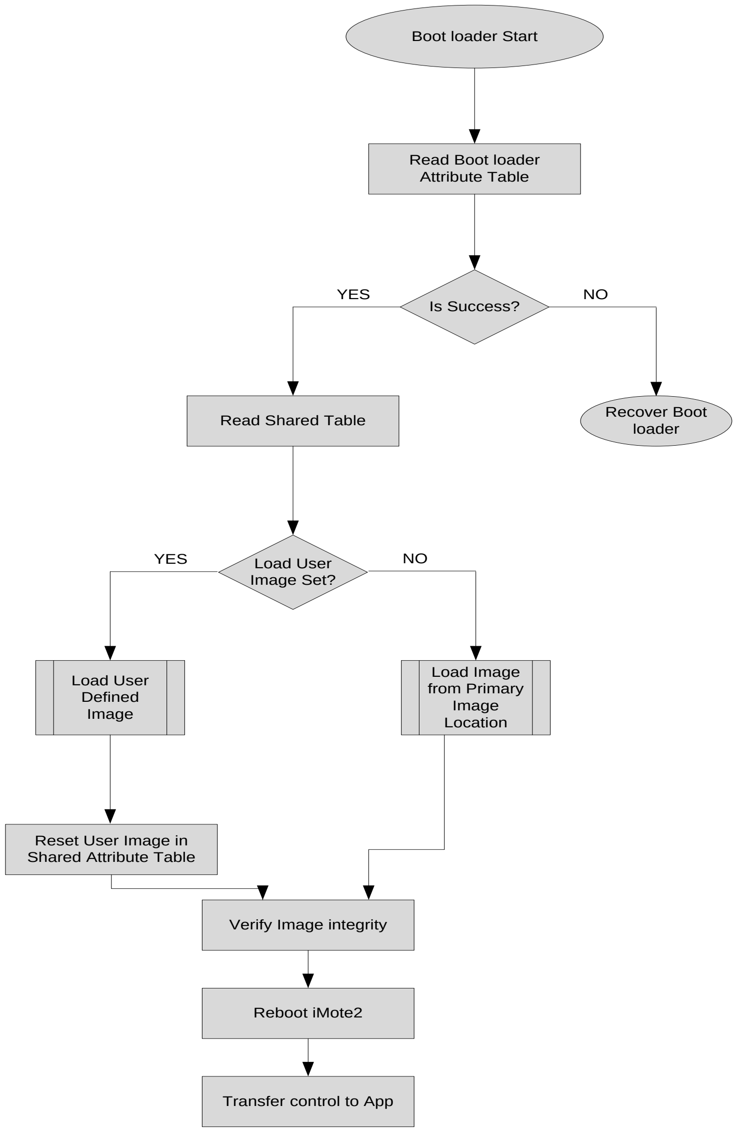


Figure17 Boot Loader Flowchart

When a new application is downloaded over the air and rebooted, we need to ensure that the TOS_LOCAL_ADDRESS and TOS_AM_GROUP which provide an identity to the node are restored. In order to achieve this, we added two additional attributes to the attribute table. So the shared attribute table is as shown in Table 1. We read these attributes and restore the node ID and group during startup in NetattrM.nc.

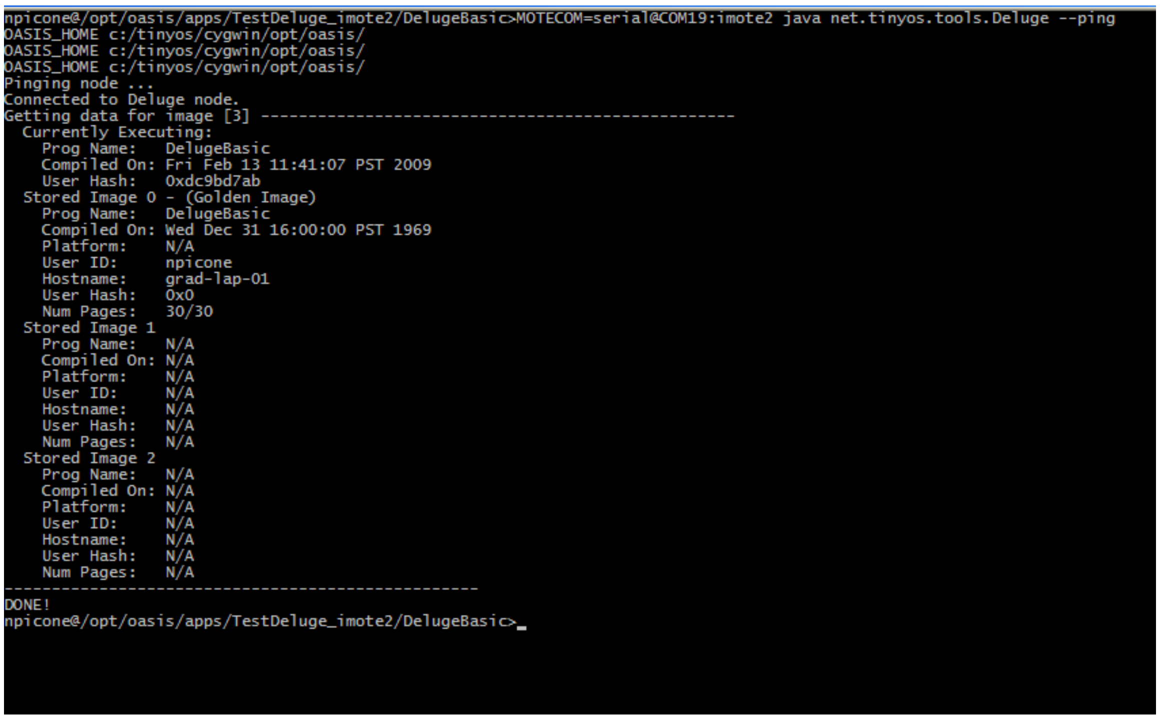
ATTRIBUTE	SIZE (Bytes)
ATTR_VERIFY_IMAGE	4
ATTR_PERFORM_SELF_TEST	4
ATTR_IMG_LOAD_LOCATION	4
ATTR_IMG_CRC	4
ATTR_IMG_SIZE	4
ATTR_SELF_TEST_IMG_LOC	4
ATTR_SELF_TEST_IMG_CRC	4
ATTR_SELF_TEST_IMG_SIZE	4
ATTR_VAR_BOOT_LOC	4
ATTR_VAR_BOOT_SIZE	4
ATTR_TOS_LOCAL_ADDR	4
ATTR_TOS_AM_GROUP_ADDR	4
ATTR_VAR_BOOT_IMG	4

Table 1 Shared Attribute Table

During the implementation, we considered scenarios that might affect OAP and accordingly added new features to ensure that OAP could handle these adverse situations. If a node restarts during the download phase, we have to ensure it starts the download again. To address this issue, we implemented a mechanism where we verify the image integrity during startup. If the image has been completely downloaded, then we continue with the normal operations. If the image has not been downloaded completely, then we erase the image downloaded and reset the metadata details. This way the node is ready to receive the image in its entirety again.

3.2.2 Implementation of GUI based Java tool chain for Deluge

We modified the Java tool that acts as an interface between the sensor network and the base station (e.g., a laptop or a desktop) to function correctly with the iMote2 platform as shown in Figure 18. The Java tool had to first compile for iMote2 platform. We had to modify the communication messages to accept and send messages with additional parameters. Finally, we had to increase the limit on the number of pages that can be downloaded per image.

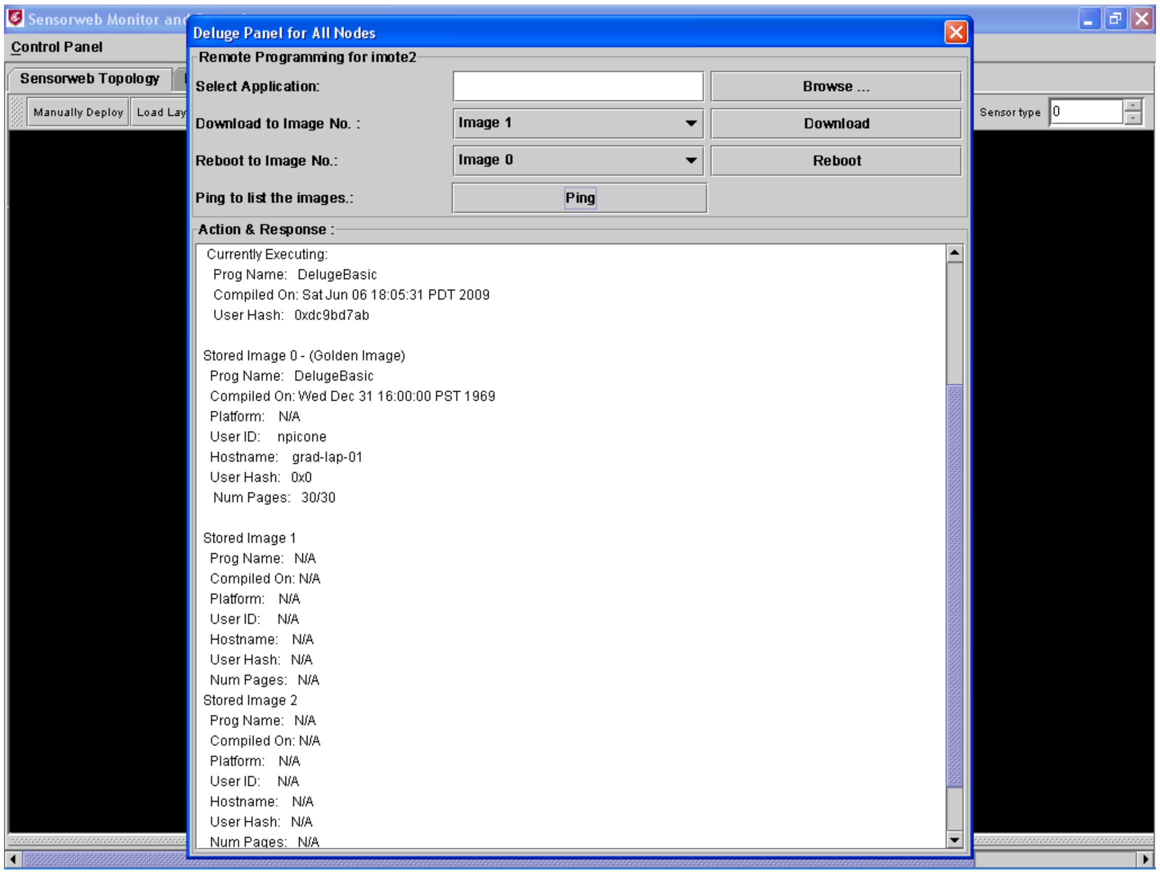


```
npicone@/opt/oasis/apps/TestDeluge_imote2/DelugeBasic>MOTECOM=serial@COM19:imote2 java net.tinyos.tools.Deluge --ping
OASIS_HOME c:/tinyos/cygwin/opt/oasis/
OASIS_HOME c:/tinyos/cygwin/opt/oasis/
OASIS_HOME c:/tinyos/cygwin/opt/oasis/
Pinging node ...
Connected to Deluge node.
Getting data for image [3] ------------------------------------------------
  Currently Executing:
    Prog Name:   DelugeBasic
    Compiled On: Fri Feb 13 11:41:07 PST 2009
    User Hash:   0xdc9bd7ab
  Stored Image 0 - (Golden Image)
    Prog Name:   DelugeBasic
    Compiled On: Wed Dec 31 16:00:00 PST 1969
    Platform:    N/A
    User ID:     npicone
    Hostname:    grad-lap-01
    User Hash:   0x0
    Num Pages:   30/30
  Stored Image 1
    Prog Name:   N/A
    Compiled On: N/A
    Platform:    N/A
    User ID:     N/A
    Hostname:    N/A
    User Hash:   N/A
    Num Pages:   N/A
  Stored Image 2
    Prog Name:   N/A
    Compiled On: N/A
    Platform:    N/A
    User ID:     N/A
    Hostname:    N/A
    User Hash:   N/A
    Num Pages:   N/A
--------------------------------------------------
DONE!
npicone@/opt/oasis/apps/TestDeluge_imote2/DelugeBasic>_
```

Figure 18Deluge Java Application

Additionally, we also developed a GUI based tool that provides an easy to use interface to disseminate the code updates. This tool allows code dissemination to any of the image slots except the Golden Image slot (which is the default). The Golden Image slot is one of the three files created in the user area. This file contains the image to be booted from if a reboot from a new image fails. This file is kept secure and cannot be overwritten by any OAP downloads. An image can only be downloaded to the Golden Image slot when the device is physically connected to the laptop/desktop via USB/UART and only using the command based Java tool.

Figure 19 shows a snapshot of the developed Deluge GUI tool. The GUI provides the same functionality as the command line tool. The GUI allows us to ping the base node, displays the different images available. It allows us to inject images into non-Golden image slots and reboot from any of the three images in the flash memory.



Select Application:
Download to Image No. : Image 1
Reboot to Image No.: Image 0
Ping to list the images.:
Sensor type 0

```
 Currently Executing:
   Prog Name:   DelugeBasic
   Compiled On: Sat Jun 06 18:05:31 PDT 2009
   User Hash:   0xdc9bd7ab

Stored Image 0 - (Golden Image)
  Prog Name:   DelugeBasic
  Compiled On: Wed Dec 31 16:00:00 PST 1969
  Platform:    N/A
  User ID:     npicone
  Hostname:    grad-lap-01
  User Hash:   0x0
   Num Pages:   30/30

Stored Image 1
  Prog Name:   N/A
  Compiled On: N/A
  Platform:    N/A
  User ID:     N/A
  Hostname:    N/A
  User Hash:   N/A
  Num Pages:   N/A
Stored Image 2
  Prog Name:   N/A
  Compiled On: N/A
  Platform:    N/A
  User ID:     N/A
  Hostname:    N/A
  User Hash:   N/A
  Num Pages:   N/A
```

Figure 19 Deluge GUI Snapshot

Every reboot copies the complete image from a location in the flash memory to bootable location. As a result, the size of the image needs to be known. Since the size of the image is currently unknown to the node at runtime, we modified the advertisement messages to include the size of the image. This allows the size of the image to be stored on the device before it is used in the reboot phase.

3.3.3Robustness Implementation

We implemented features to ensure that Deluge is robust. We describe the different shortcomings of Deluge implementation for iMote2 and the solution implemented to overcome the same. The download is based on version numbers. The version number is an 8-bit integer. Therefore, a maximum of 255 downloads are possible throughout the lifetime of the network. This number is significantly small if the network has a large lifespan. As a result, we modified this variable to be a 32-bit integer. To add more robustness to the version number, instead of using a counter that increments once every download, we use UNIX time stamp to be the version number. Now, the user will have the knowledge of the version number currently running on all the nodes just based on the compilation time of the application.

As mentioned previously, the metadata table has a limited size. The size of the table is sufficient to allow two images to be downloaded completely without rebooting. However, if a user downloads more than two images continuously, the metadata table overflows and the node dies. The node can only be restarted by manually plugging out and plugging back in the connection to iMote2. To make Deluge robust to this type of failure, we designed and implemented the solution shown below in Figure 20.

To ensure that the Java tool running on the PC identified the messages with Unix timestamp as the version number, we modified the Java tool to do some book keeping of information related to the version numbers. We will present the evaluation of all the solutions proposed in the next chapter.

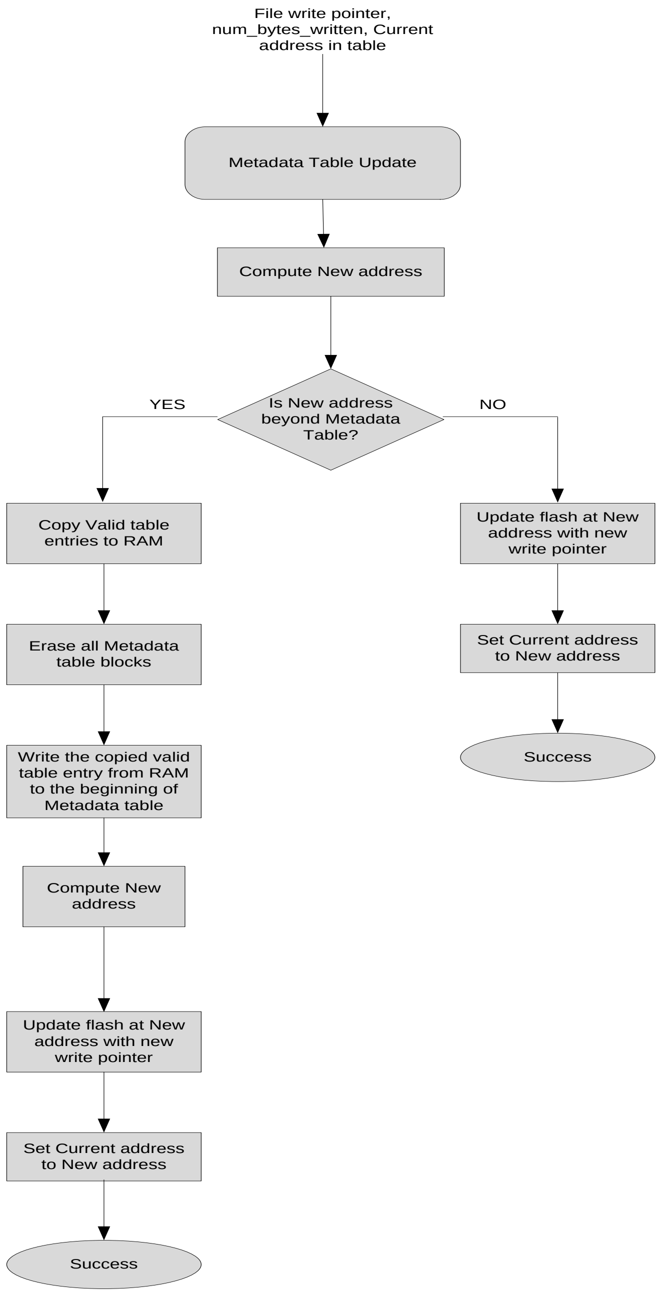

File write pointer, num_bytes_written, Current address in table
Metadata Table Update
Compute New address
Is New address beyond Metadata Table?
YES
NO
Copy Valid table entries to RAM
Erase all Metadata table blocks
Write the copied valid table entry from RAM to the beginning of Metadata table
Compute New address
Update flash at New address with new write pointer
Set Current address to New address
Success
Update flash at New address with new write pointer
Set Current address to New address
Success

Figure 20 Robust Metadata Table Update

CHAPTER FOUR

PERFORMANCE EVALUTION AND ANALYSIS

We evaluated our OAP protocol using the following metrics: (a) Completion time, which calculates the time necessary for dissemination of the binary object; (b) Download time for an individual node through the USB; and (c) Number of different messages exchanged in the network because of the OAP process. Further more we evaluated the robustness of the system based on (a) Version numbers (b) Multiple downloads.

4.1 Laboratory Setup and Results

4.1.1 Deluge

We evaluated our Deluge implementation on a laboratory network of iMote2. We used a grid network as shown in Figure 21 and evaluated OAP performance on networks of different sizes (from 2 to 24 sensor nodes). Evaluation of our protocol on TOSSIM (a simulator for WSN in TinyOS) was not possible because TOSSIM is not compatible with iMote2 platform. TOSSIM does not emulate the hardware functionalities of iMote2.

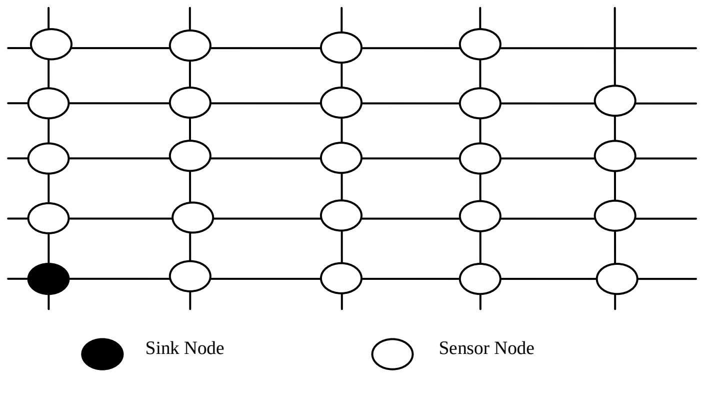


Figure 21 WSN Structure

We ran each experiment 10 times and computed the average time for completion. The radio frequency power was set to 2 for each node (ensuring that a node can only communicate with its immediate neighbors).

4.1.1.1 Completion Time

Completion time was evaluated for different size applications. We classified the size of the images as small, medium and large. A TinyOS application which just prints a debug statement was created and considered small application, where as our project application [31] called OasisApp, which senses seismic data and routes the data back to the sink, was considered a medium application. OasisApp with many additional components included, was considered as a large sized application. The sizes of the applications are as shown in Table 2.

Type	**Application**	**Size (in KB)**
Small	Test Application	126
Medium	OasisApp	237
Large	LargeTestApp	544

Table 2 Application Type

As a baseline for comparison, we also calculated the time it takes to download an application onto an iMote2 sensor through the UART/USB. The boot loader console, which communicates with the mote, divides an application into equal sized chunks and sends the data to the mote accordingly. The average download time for UART/USB-based download of a small size or a medium size application is about 26 to 30 seconds and about 45 to 50 seconds for a large-sized application. We also need to include the time the PC takes to recognize the device once it is plugged in to the USB port. This takes approximately 40 seconds. Therefore, it takes approximately 1:08 minutes to download small or medium sized application and approximately 1:30 minutes to download a large-size application on to a single iMote2 sensor node via the USB port. To program 24 nodes with small or medium size applications, it takes approximately 26 minutes and for a large-size application it takes approximately 36 minutes. Hence, over the air programming generally takes less time and is much easier and less labor-intensive for disseminating the binary applications.

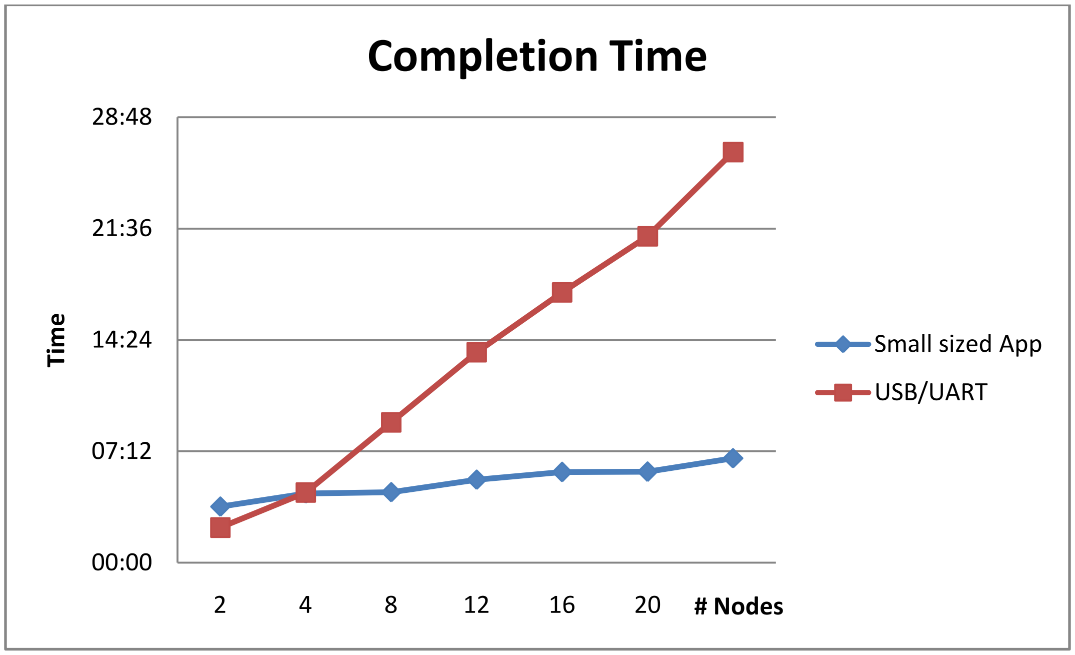


Figure 22 Completion Time Small Sized App

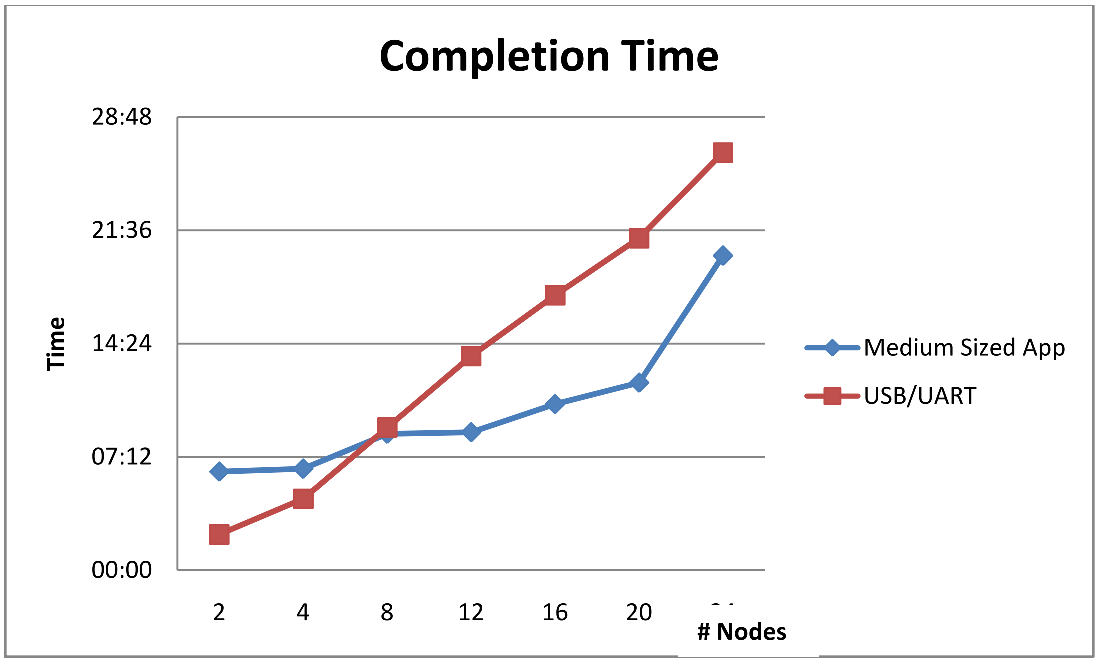


Figure 23 Completion Time Medium Sized App

Figure 22, Figure 23 and Figure 24 respectively show the completion times for small, medium, and large size applications and compare them to the time it takes to download the same application via the USB/UART port.

As demonstrated in Figures 21-23, OAP significantly outperforms reprogramming of sensor nodes via the USB port when the number of nodes goes beyond 4, 8 and 16 respectively. For example, it takes more than 3 times as long to reprogram 24 sensor nodes via the USB port as opposed to OAP for small programs. For large programs, we can see that OAP outperforms reprogramming via the USB port when the number of nodes is greater than 16.

When using the USB port for small and medium applications, the overhead is dominated by the lag time it takes to recognize a sensor node once it is plugged into the port, thus, it is clear that OAP yields better timing advantages, even for smaller networks, compared to USB-based reprogramming for smaller to medium size applications by eliminating this lag time.

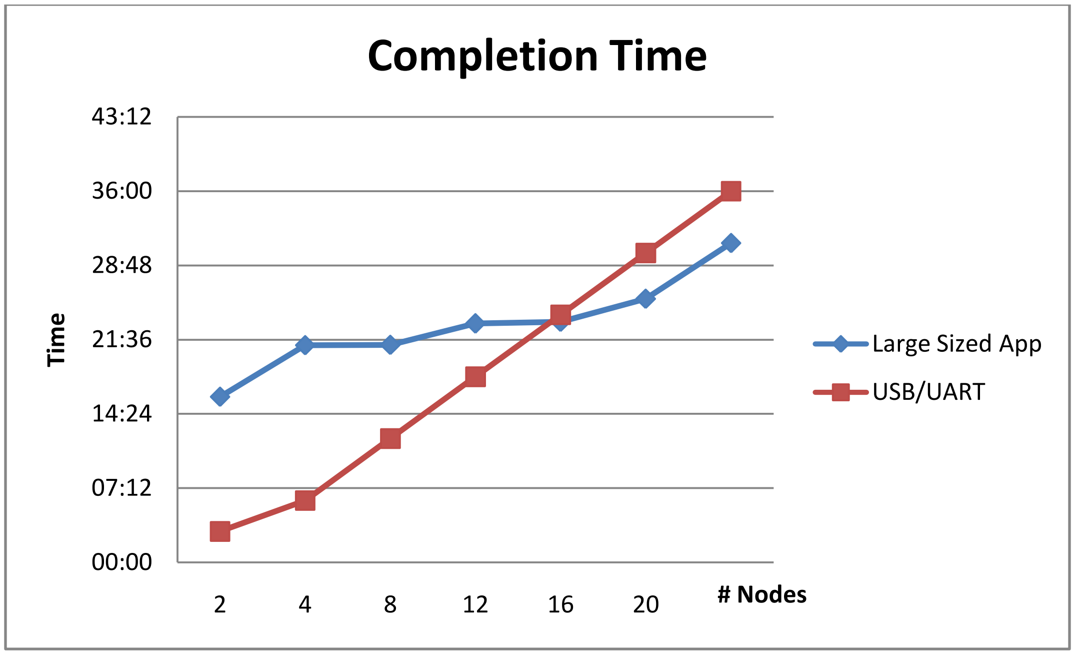


Figure 24 Completion Time Large Sized App

On the other hand, using OAP for large programs, the communication delay for uploading the large chunks of the code becomes dominant. In comparison, fast USB connection reduces the communication delay for uploading a program using the USB port. Thus, in the case of large programs, it takes a larger network to realize advantages of the OAP by overcoming the constant lag time for recognizing USB devices. Note that we were not able to use more sensor nodes due to the limited size of our test bed WSN. We believe that much more significant performance advantages would be attained for large programs when larger networks are deployed.

4.1.1.2 Message Overhead

As a measure of the overhead of the protocol, we evaluated the number of messages being transmitted during the entire dissemination. This is measured as the ratio of number of protocol packets over the number of data packets transmitted (a smaller ratio indicates a smaller

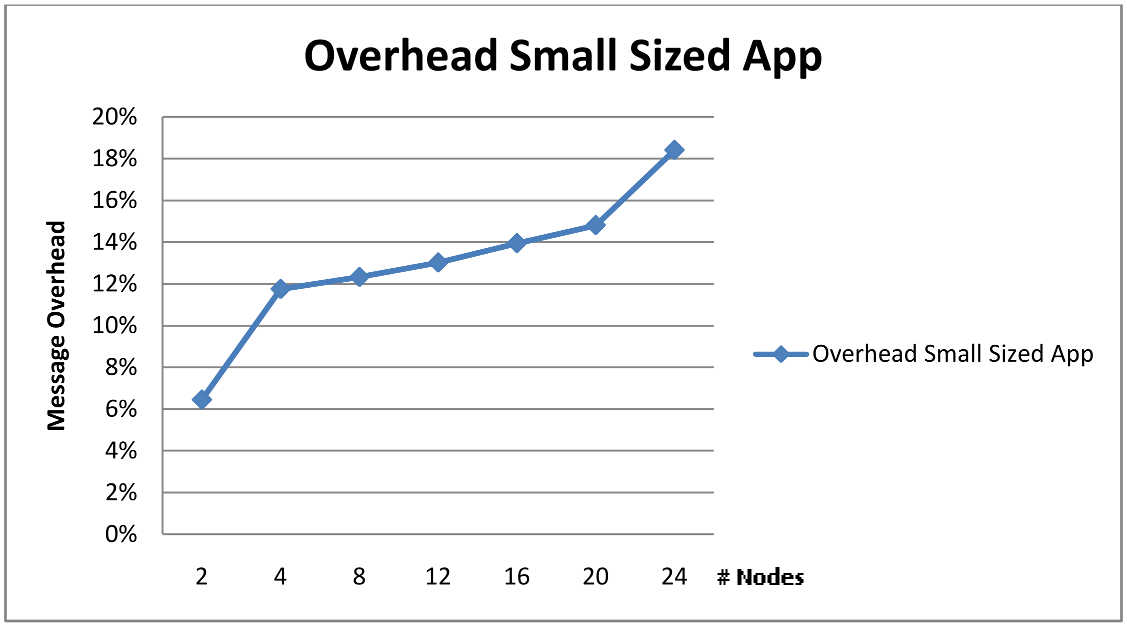


Figure 25 Message Overhead Small Sized App

overhead). By protocol packets we mean the request messages (Req) and the advertisement messages (Adv) transmitted during the OAP process.

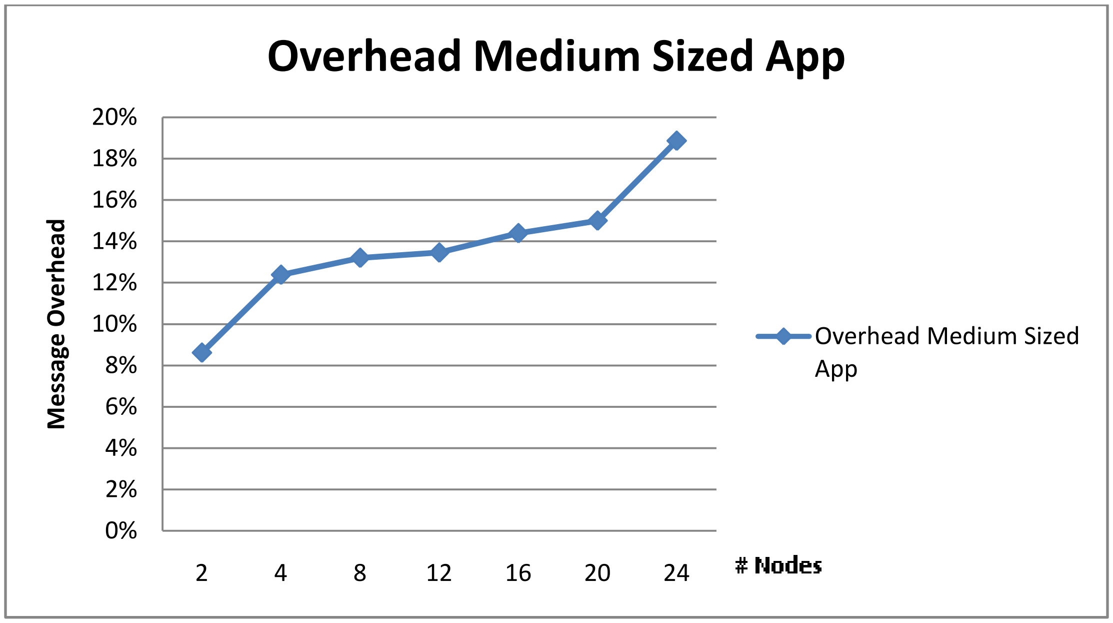


Figure 26 Message Overhead Medium Sized App

We calculated the number of advertisement messages, the number of request messages and the number of data messages exchanged in the network for each type of application.

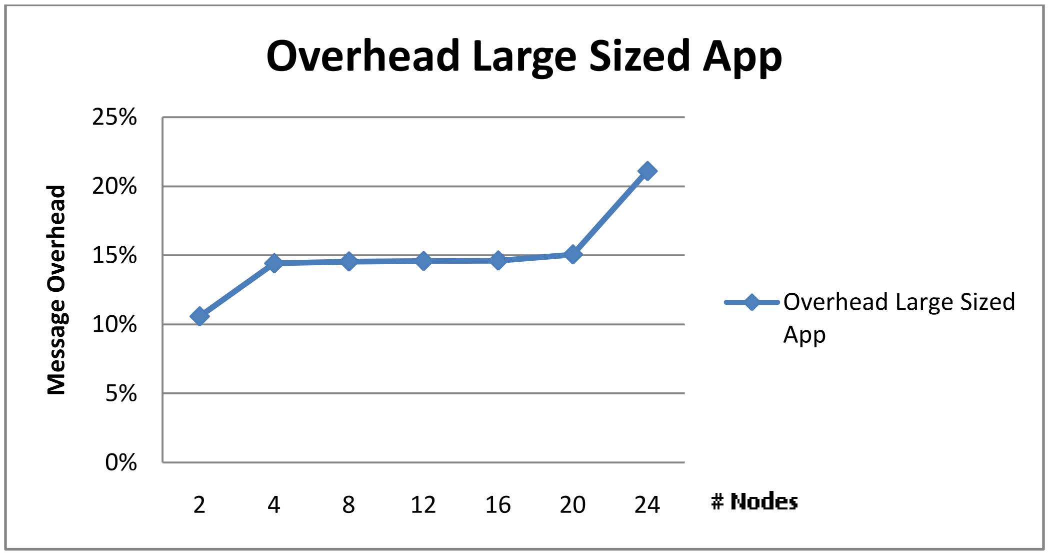


Figure 27Message Overhead Large Sized App

As seen in Figures 24-26 the message overhead associated with OAP remains small even for large sized applications. For small and medium size applications the message overhead is below 20 percent even for a large number of nodes. When the size of the application is significantly increased, the message overhead still remains small at less than 21 percent. We believe that the slight increase in the message overhead for larger applications is due to the number of pages being transmitted. When a page is being downloaded, the node cannot respond to request or acknowledgement messages, thus they must be resent. Therefore, as the number of pages increases, it causes a slight increase in the overhead messages that need to be retransmitted.

Deluge can execute concurrently with some applications in order to keep the state information needed to carry out the OAP protocol. Note that iMote2 sensors have a limited amount of RAM space for programs to execute (maximum of 256 KB). Our Deluge protocol implementation is very efficient and only requires 48KB of RAM space to execute, leaving plenty of space for the other application.

4.1.1.3WSN Deployment on Mt. St. Helen

Finally, it is worth mentioning that we deployed 15 nodes on mount St. Helen during the summer of 2009. The network was enabled with Deluge and we reprogrammed a partial network twice. We reprogrammed two nodes out of the 15 nodes the first time. The reprogramming took about 5.5 hours to complete (due to the distance) and the reprogramming was successful. Both the nodes rebooted successfully with the new binary image. However, when we reprogrammed another node separately, the node did not reboot successfully. The reason for this could be that Deluge is based on gossip protocol and the gossip protocol does not work efficiently in a sparse network [32]. We would like to further test OAP in the field rigorously and overcome the drawback with respect to gossip protocol.

4.1.2 Robust Deluge

We evaluated the robustness implementation discussed in Chapter 3 on a laboratory network of iMote2. We used a grid network as shown in Figure 21 and evaluated success rate of reprogramming on networks of two different sizes (6 and 12 sensor nodes). We ran the experiment 3 times each and computed the percentage for successful download. The radio frequency power was set to 2 for each node like the previous test cases.

4.1.2.1 Evaluation based on Version Number

Robustness was evaluated for different sized applications mentioned in Table 2. We classified the network as having images with random version numbers, sequential version numbers, same version numbers and no images at all. By sequential version numbers we mean the compilation time of each image increases sequentially. For example, node A has an image in slot 1 with version number 0x4AF9FA2F (Standard Time: 11/10/2009 23:41:35 GMT) and node B has the same image(on slot 1) with compilation time as 0x4AF9FA30 (Standard Time: 11/11/2009 23:41:36 GMT). By random version numbers we mean that the compilation time of each image is random and are not necessarily sequential. For each setup, we tested the network by downloading a new application with a new and higher version number.

Table 3 and Table 4 show the results for the test cases where the entire network is running with the same version number and with no image respectively. For the three runs that we conducted, all the nodes in the network successfully downloaded the new version image into one of the slots for the two different sized networks.

Test Scenario	Application Type	Run 1 % Success		Run 2 % Success		Run 3 % Success	
# Nodes		6	12	6	12	6	12
Same Version	Small	100	100	100	100	100	100
	Medium	100	100	100	100	100	100
	Large	100	100	100	100	100	100

Table 3 Same Version Number in the network

Table 5 and Table 6 show the results for the network with sequential version numbers and random version numbers.

Test Scenario	Application Type	Run 1 % Success		Run 2 % Success		Run 3 % Success	
# Nodes		6	12	6	12	6	12
No Images	Small	100	100	100	100	100	100
	Medium	100	100	100	100	100	100
	Large	100	100	100	100	100	100

Table 4 No Images in the network

The success rate for a network with sequential version numbers was found to be 100 percent for both 6 and 12 node network. However, we found that while reprogramming the network having random versions (6 node network) with medium and large sized images, the success rate was not 100 percent. During the download phase for medium sized application, the cable providing power to a node was unplugged inadvertently; the node did not download successfully at the first time.

Test Scenario	Application Type	Run 1 % Success		Run 2 % Success		Run 3 % Success	
# Nodes		6	12	6	12	6	12
Sequential Versions	Small	100	100	100	100	100	100
	Medium	100	100	100	100	100	100
	Large	100	100	100	100	100	100

Table 5 Sequential Versions in the Network

After plugging the power cable to the node, the device successfully rebooted and downloaded the application. While reprogramming the network with a large sized application, a couple of nodes were placed at a corner and were not surrounded by enough neighbors. As a result, they did not successfully download on the first attempt. Once the nodes were moved closer to other nodes, they reprogrammed successfully with the new image.

Test Scenario	Application Type	Run 1 % Success		Run 2 % Success		Run 3 % Success	
# Nodes		6	12	6	12	6	12
Random Versions	Small	100	100	100	100	100	100
	Medium	100	100	100	100	84	100
	Large	100	100	100	100	67	100

Table 6 Random Versions in the network

4.1.2.2 Multiple Downloads

We further evaluated our robustness implementation by downloading images continuously without rebooting the nodes. We performed this stress test to ensure that the solution developed to overcome the metadata table overflow was robust. Table 7 provides our evaluation result for multiple downloads on both 6 and 12 node network. We again reprogrammed the network with different sized applications mentioned in Table 2.

Test Scenario	Application Type	Run 1 % Success		Run 2 % Success		Run 3 % Success	
# Nodes		6	12	6	12	6	12
Multiple Downloads	Small	100	100	100	100	100	100
	Medium	100	100	100	100	100	100
	Large	100	100	100	100	100	100

Table 7 Multiple Downloads

From the results above we see that multiple downloads for small, medium and large sized applications were 100 percent successful. We encountered an unexpected problem while evaluating the robustness. The download proceeds smoothly when the metadata table is not full. Once the metadata table is about to overflow, we copy all the valid entries from the table to RAM and erase all the blocks allocated for the table. The erase function call is a blocking function call, in that the processor cannot accept any other function call while the erase function is executing. Once the node erases the table and rewrites the entries into the table, the node ceases to function normally. The nodes are unable to receive or send any data packets. However, they are able to receive advertisement. The nodes cease to request data packets because each node suppresses sending request messages. As a result the nodes are unable to communicate with each other to send and receive data packets. We fixed this problem by ensuring that at least the base node does not suppress request messages. For evaluation purpose, we ran the experiment 3 times and only tabulated the results for each application.

After analysis of the evaluation results, we believe that the implementation of Deluge for iMote2 platform is a significant step forward to achieve the required flexibility and robustness for WSN. Primarily, the design and implementation of the functionalities of Deluge has allowed us to successfully reprogram a network of iMote2 deployed in remote areas. Secondly, by making Deluge robust to node failure, we have ensured that the performance of the protocol is improved.

CHAPTER FIVE

CONCLUSIONS

In this work, we successfully implemented and tested our dynamic over the air programming protocol in a resource-constrained environment and also ensured that it is robust. Our goal was to enable over the air programming for iMote2 sensors. We accomplished this by modifying Deluge so that it writes and reads data onto the flash memory and reboots from any location. In order to evaluate the protocol we downloaded images of different sizes on networks of varying size and were able to successfully reboot using each of the different applications. Our results from a real controlled test bed have shown that Deluge does indeed work for iMote2 platform efficiently. We were able to define the different problems that we had to deal with to make Deluge functional. We also proved that Deluge works well in a field deployment.

We further ensured that Deluge is robust enough to both hardware and software failure by developing some approaches like the protocol does not fail if the number of updates during the lifetime exceeds 255 and the number of updates is based on the UNIX time stamp. The 32-bit integer overflows only in 2038. Therefore, our protocol can be functional until that time. We further ensured that the protocol does not fail if multiple updates are provided due to Metadata table overflow. We added some features to the table so that the nodes continue to function in spite of many consecutive downloads.

We finally developed a GUI based Java tool to enable user friendly interactions during the download process. This tool can be used as a standalone application or can be used with other monitoring tools developed for our project. The ease of use and accessibility has made this a good application for our project.

CHAPTER SIX

FUTURE WORK

The areas that we worked on can be further developed to achieve better performance. In this chapter we discuss the various improvements on Deluge and robustness aspects and also introduce the security aspect, which is of prime importance to us. We provide a basic authentication algorithm we designed and also discuss the advantages of the approach.

Deluge is an efficient protocol. However, it functions efficiently in lab environment but is not effective in field because Deluge is based on gossip protocol. There are well known drawbacks of gossip protocol like gossip nature decreases as the number of neighbors' decreases. We can overcome this drawback in Deluge by using other mechanisms like using a tree based multicast communication instead of broadcast communication as proposed by Pao et. al. [32].

Deluge is robust to hardware and software failures to a large extent. However, a node might fail to successfully startup if the node restarts while reading or writing some partitions of the flash memory. During startup of the node, the sector table is read into memory. During this read, if the node fails due to reasons like power outage, then there is no guarantee that Deluge will startup successfully. The same is the case with the attribute tables. Before reboot, we write certain values to attribute tables like the address of the image, the size of the image, the TOS_LOCAL_ADDRESS and TOS_AM_GROUP. During this time, if the node fails due to some reason, there is no guarantee that the node starts up successfully. The final situation where there is no guarantee for a successful startup would be with respect to the metadata table. The metadata table gets updated during binary image download. While updating this table a node restart might not ensure a successful startup of Deluge. These situations can be handled by implementing a back up table which will ensure a successful recovery of the device. Also, the corruption of bit vector during metadata table erase should be investigated further and a solution to prevent this corruption should be implemented.

Security is another dimension of robustness and is of utmost importance to us. As the network is deployed in the field, the probability of an adversary to eavesdrop on the communication and download malicious code increases.

The OAPs developed do not support security. Unfortunately, security mechanisms like public key cryptography and other mechanisms were not compatible with the resource constrained devices like sensor nodes until recently. As a result, security remained a neglected aspect of OAP for sometime. Development of the cryptographic libraries for resource constrained devices like TinyECC [33], TinySec [34] and MiniSec [35] have helped develop strong and secure methods of security and authentication for OAP.

Some of the most common weaknesses of OAP in WSN are (i) It is easy for an adversary to inject bogus packets into the network as OAP is based on broadcast medium of communication. Such an attack cannot be detected before an entire image has been downloaded. (ii) The nodes need to buffer the program in the external memory before checking its integrity and authenticating it, as RAM space is not sufficient to hold the entire image. However, writing the data to the external flash is an expensive operation. Various technologies are available to erase sectors or blocks of memory and relocating the image to different blocks. But, this only increases the cost of verification and integrity check. (iii) Such attacks have a large impact on the receiver node in terms of energy consumed for receiving, storing and verifying while it might be very less for the attacker.

Most authentication techniques use the standard encryption algorithms available like those mentioned in [36], [37], [38] and [39]. Many security mechanisms have been developed for securing Deluge and other OAP protocols. Perriget. al. [40] proposed an authenticated broadcast protocol called μTESLA. However it is inadequate as it is a purely cryptographic protocols and does not propose an energy efficient means to disseminate data in the network. Sluice [41] applies authentication over Deluge protocol. However, Sluice places some strict ordering on the system like in-order delivery of packets, verification after buffering the whole packet and download of $i+1^{th}$ page only after i^{th} page has completed. Due to these restrictions, too many resources are being used and it is not an efficient mechanism. Subsequently Dutta et.al [42] proposed a packet level data authentication mechanism. Deng et.al. [43] proposed a hash chain that is received in advance to enforce the packet to be received out of order. However, using only one hash for all the packets makes the system vulnerable to DoS attack. Hence the authors use Merkle hash tree to over come the problem. A patented proposal by Waltro et al [44] provides a public key scheme to secure the advertisement and request messages. However the drawback is the requirement of resources.

Sokjoonet. al. [45] Proposed a PKI based digital signature to authenticate the advertisement packets. Das et al [46] propose a data dissemination mechanism based on the mathematical principle of orthogonality. Mathematically, two vectors are orthogonal if their dot product is zero. This principle is used in many areas like TDMA, FDMA and CDMA. Given one vector, we can find infinite number of vectors that are orthogonal to the given vector over the given vector space and they all have to be mutually orthogonal. Osman Ugus et al [47] proposed a secure code update mechanism considering the resource constraint of the sensor devices. As most of the techniques use expensive energy and space consuming, the authors came up with a stateful-verifier signature built around Merkle one time signatures.

The latest technique developed for securing OAP is the one proposed by Hailun Tan et. al [48]. This technique provides authentication of the messages in addition to securing the communication. This approach uses all the state-of-the-art methods to secure and authenticate communication. The authors propose to use hash trees, one way hash chains and digital signature and also a cipher puzzle. They also provide solutions to overcome signature based and request based DoS attacks.

The techniques developed so far are computationally intensive in that they consume time, space and energy for execution. Also, the algorithms developed make the protocol too complicated. The amount of security being included would be applicable to military applications. However, for applications like monitoring an environment, these techniques seem to bean overkill. As a result we propose a simple yet robust authentication technique to authenticate the communication messages. Before explaining the protocol, we first give a brief description of the threat model.

As our network is deployed on Mount St. Helen, physical access to nodes is not easy. Therefore, we are only concerned about the communication authentication. We assume that the adversary has access to powerful computational capabilities such as a laptop. The adversary is capable of launching external attacks. No valid node is physically compromised and the adversary can eavesdrop, copy and replay the messages. Insider attack like physical compromise of a node is not addressed in this algorithm. The base station cannot be compromised and it has unlimited computational power compared to other sensor nodes. The sensor nodes can perform limited number of cryptographic operations. We also assume that the packet size is large enough to hold the cryptographic data.

There are only three different types of messages that require authentication in Deluge namely, Advertisement, Request and Data messages. Our application does not require high-end security. As the TinyECC library and digital signatures are expensive, we propose to use symmetric key encryption. MiniSec [35] is a cryptographic library readily available as part of TinyOS. MiniSec provides authentication as well as encryption of messages. This library is compatible with Tmote Sky. As Tmote Sky and iMote2 use the same radio CC2420 hardware, the library can be easily ported to iMote2 platform. We propose to use MiniSec to provide the symmetric key encryption for Deluge. We will generate a 128-bit master key and a 128-bit Binary ID and preload it to all the nodes in the network. The nodes encrypt the binary ID using the master and attach it to every message sent. During network startup, all the nodes can communicate with each other using the encrypted advertisement message.

To ensure that the master key does not become obsolete, we also propose to use a version key for every version downloaded in the network. The version key can be generated dynamically at the base node and can be propagated in the network by encrypting the version key using the master key. So every binary image download is encrypted and authenticated using a new version key. The pseudo code is presented below. When a node receives an advertisement message, it will verify the authenticity of the sender by decrypting the 128-bit binary ID using the master key. If the decryption is successful and the version number is greater than the existing version number, decrypt the new version key to be used for image download and store, and setup the slot to download the image. If the decryption was not successful, ignore the message and increment the counter for the sender message. If the counter value reaches a threshold value, then flag the node as an adversary and drop all the packets from the node. Request message is authenticated along the same lines.

```
ADVERTISEMENT MESSAGE
If Receive Advertisement message
  Verify Authenticity of the sender (verify the 128-bit binary ID)    If Authentic
        If version number is more than the existing version number
        Decrypt the new version key using the master key. Store the version key for future use.
        Setup the slot for image download
        Send Request message (after encrypting the binary ID using the new version key)
        End if (version number)
  Else
        Ignore message, increment counter for the sender node
        If counter for a sender node = threshold
              Report adversary
        End if
  End if (authentic)
End if (Receive Advertisement message)
```

Figure 28 Advertisement Message Authentication

```
If Send Advertisement message
  Encrypt the binary ID stored in flash using the master key.
  Attach the version key stored after encrypting using the master key (Note: The nodes are not generating this key but the java tool will generate this key. So computational overhead is reduced)
 Send message
```

Figure 29 Encryption of Advertisement Message

When a node sends an advertisement message periodically, it attaches the binary ID that is encrypted using the master key along with other metadata information.

```
REQUEST MESSAGE
If received Request Message
    Verify Authenticity of the sender [Decrypt the binary ID using the new version key]
    If Authentic
        Encrypt the binary ID using the new version key
        Attach the version key to the data packet.
        Send Data message
    Else
        Ignore message
        Increment the counter for the sender.
    End if
End if (request message)
```

Figure 30 Request Message Authentication

```
If send Request Message
    Encrypt the binary ID using the new version key
    Attach the version key to the message
  Send request message
End (send Request Message)
```

Figure 31 Encryption of Request Message

When a node sends a request message, it encrypts the binary ID using the new version key sent in the advertisement. This way the receiver node proves to the sender node that it successfully decrypted the message.

```
DATA MESSAGE
If received Data Message
  Verify Authenticity of the sender [Decrypt the binary ID using the version key]
 If Authentic
    Write data packets to flash (without the key)
  Else
     Drop packets Increment the counter for the sender
  End if
End if (data message)
```

Figure 32 Data Message Authentication

All the data packets are authenticated using the new version key. If the decryption is successful, write the data packets to flash. If the decryption is not successful, drop the packet and increment the counter for the sender.

```
If send Data Message
   Encrypt the binary ID with the new version key
   Attach the encrypted value to the data packets
  Send Data message
End Send Data Message
```

Figure 33 Encryption of Data Message

The data packets are sent after attaching the binary ID encrypted using the new version key.

This algorithm is a basic idea of the authentication protocol that can be developed. We also think that functionality to change the keys periodically can be considered. Cryptanalysis of this protocol needs to be done.

CHAPTER SEVEN

BIBLIOGRAPHY

1. **Wikipedia.** *http://en.wikipedia.org/wiki/Over-the-air_programming.*

2. *The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale.* **Culler, J. W. Hui and D.** s.l. : Proc. ACM SenSys, 2004.

3. *Wireless Sensor Networks for Habitat Monitoring.* **Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, John Anderson.** s.l. : ACM, 2002.

4. *Wireless Sensor Networks for Battlefield Surveillance.* **Tatiana Bokareva, Wen Hu, Salil Kanhere, Branko Ristic, Neil Gordon, Travis Bessell, Mark Rutten, Sanjay Jha.** Brisbane : Proceedings of the Land Warfare Conference, 2006.

5. *An Advanced Wireless Sensor Network for Health Monitoring.* **G. Virone, A. Wood, L. Selavo, Q. Cao, L. Fang, T. Doan, Z. He, R. Stoleru, S. Lin, J.A. Stankovic.** s.l. : Mobile and Ubiquitous Systems: Networking and Services, 2005, 2005.

6. *Ad-Hoc Wireless Sensor Networks For Exploration Of Solar-System Bodies.* **Philippe Dubois, Cyril Botteron, Valentin Mitev, Carlo Menon, Pierre-André Farine, Paolo Dainesi, Adrian Ionescu, Herbert Shea.** s.l. : Acta Astronautica, 2009.

7. *Reprogramming Wireless Sensor Networks:Challenges and Approaches.* **Qiang Wang, Yaoyao Zhu, and Liang Cheng.**

8. *A Remote Code Update Mechanism for Wireless Sensor Networks.* **T. Stathopoulos, J. Heidemann, and D. Estrin.** Tech. rep. CENS-TR-30, UCLA : Center for Embedded Networked Computing, Nov. 2003.

9. *MNP: Multihop Network Reprogramming Service for Sensor Networks.* **S. S. Kulkarni, and L. Wang.** s.l. : Proc. IEEE ICDCS , 2005.

10. *Optimized Autonomous Space In-situ Sensor-Web for Volcano Monitoring.* **Song, W.Z. and Shirazi, B. and Kedar, S. and Chien, S. and Webb, F. and Tran, D. and Davis, A. and Pieri, D. and LaHusen, R. and Pallister, J.** s.l. : IEEE Aerospace Conference, 2008.

11. *http://sensorweb.vancouver.wsu.edu/people.html.* **Wiki, SensorWeb.**

12. **Nina Peterson, Lohith Anusuya-Rangappa, Behrooz Shirazi, Renjie Huang, Wen-Zhan Song, M. Miceli, D. Mcbride, Ali Hurson, and Rick LaHusen.** *TinyOS-based QoS Management in Wireless Sensor Networks.* s.l. : Hawaii International Conference on System Sciences (HICSS), 2009.

13. **Wen-Zhan Song, Renjie Huang, Behrooz Shirazi, Rick LaHusen.** *TreeMAC: Localized TDMA MAC Protocol for High-throughput and Fairness in Sensor Networks.* s.l. : Seventh Annual IEEE International Conference on Pervasive Computing and Communications (PERCOM), 2009.

14. *http://www.cs.uvm.edu/~crobinso/mote/tmote-sky-datasheet-102.pdf.* **Crossbow Technology, Inc.** 2003.

15. *Mote In-Network Programming User Reference and Mica2 Wireless Measurement System Datasheet.* **Crossbow Technology, Inc.** 2003.

16. *Fidelity and Yield in a Volcano Monitoring Sensor Network.* **Geoff WernerAllen, Konrad Lorincz, Jeff Johnson, Jonathan Lees, Matt Welsh.** s.l. : Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation , 2006.

17. *Rateless Deluge:Over-the-Air Programming of Wireless Sensor Networks using Random Linear Codes.* **Andrew Hagedorn, David Starobinski, and Ari Trachtenberg.** s.l. : IPSN, 2008.

18. *Mote In-Network Programming User Reference.* 2003. p. 8.

19. *Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks.* **al., P. Levis et.** s.l. : Proc. 1st Symp. Networked Sys. Design and Implementation, 2004.

20. *The Firecracker Protocol.* **Culler, P. Levis and D.** Leuven, Belgium : Proc. 11th ACM SIGOPS, Sept. 2004.

21. *Sprinkler: A Reliable and Energy Efficient Data Dissemination Service for Wireless Embedded Devices.* **al, V. Naik et.** s.l. : 26th IEEE Real-Time Sys.Symp, Dec. 2005.

22. *Aqueduct: Robust and Efficient Code Propagation in Heterogeneous Wireless Sensor Networks.* **Phillips, L. A.** s.l. : Master's thesis, Univ. CO,, 2005.

23. *Astrolabe: A Robust and Scalable Technology For Distributed System Monitoring, Management, and Data Mining.* **Robbert Van Renesse, Kenneth P. Birman, Werner Vogels.** s.l. : ACM Transactions on Computer Systems (TOCS), 2003. pp. 164 - 206.

24. *The Ninja architecture for robust internet-scale systems and services.* **Steven D. Gribble, Matt Welsh , Rob Von Behren , Eric A. Brewer , David Culler , N. Borisov , S. Czerwinski , R. Gummadi , J. Hill , A. Joseph , R. H. Katz , Z. M. Mao , S. Ross , B. Zhao.** s.l. : Computer Networks: The International Journal of Computer and Telecommunications Networking, 2001. pp. 473 - 497.

25. *http://docs.tinyos.net/index.php/Main_Page.* **TinyOS.**

26. *The nesC language: A holistic approach to networked embedded systems.* **Gay, D. and Levis, P. and Von Behren, R. and Welsh, M. and Brewer, E. and Culler, D.** NewYork : ACM Sigplan Notices, 2003. Vol. 38.

27. *http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/Imote2_Datasheet.pdf.* **Crossbow.**

28. *http://www.xbow.com/Products/productdetails.aspx?sid=253.* **Crossbow.**

29. *http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/IIB2400_Datasheet.pdf.* **Intel.**

30. *Over the Air Programming on Imote2-based Sensor Networks.* **Rashmi Parthasarathy, Nina Peterson, Behrooz A. Shirazi, WenZhan Song, Ali Hurson.** Hawaii : HICSS, 2010.

31. *Optimized Autonomous Space In-Situ Sensorweb, https://sensorweb.vancouver.wsu.edu/wiki/index.php/Main_Page.* **WSU.**

32. *Tree-Based versus Gossip-Based Reliable Multicast in Wireless Ad Hoc Networks.* **Pao, K. S. Lau and Derek.** s.l. : IEEE CCNC 2006 proceedings, 2006.

33. *TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks.* **An Liu, Peng Ning.** s.l. : 7th International Conference on Information Processing in Sensor Networks, 2008. pp. 245-256.

34. *TinySec: a link layer security architecture for wireless sensor networks.* **Chris Karlof, Naveen Sastry and David Wagner.** s.l. : Proceedings of the 2nd international conference on Embedded networked sensor systems, 2004. pp. 162 - 175 .

35. *MiniSec: a secure sensor network communication architecture.* **Mark Luk, Ghita Mezzour, Adrian Perrig, Virgil Gligor.** s.l. : Proceedings of the 6th international conference on Information processing in sensor networks, 2007. pp. 479-488.

36. **Schneier, Bruce.** *Applied Cryptography.* s.l. : John Wiley & Sons, 1996.

37. **Charlie Kaufman, Radia Perlman, Mike Speciner.** *Network Security: Private Communication in a Public World.* 2002.

38. **Stallings, William.** *Cryptography and Network Security: Principles and Practice.* s.l. : Pearson Edition, 2006.

39. **Spillman, Richard J.** *Classical and Contemporary Cryptology.* s.l. : Prentice Hall, 2005.

40. *SPINS: Security Protocols for Sensor Networks.* **A. Perrig, R. Szewczyk, V. Wen, D. Culler and J.D. Tygar.** s.l. : Proceedings of the 7th Ann. Int. Conf. on Mobile Computing and Networking, 2001. pp. 189-199.

41. *Sluice: Secure Dissemination of Code Updates in Sensor Networks.* **P.E. Lanigan, R. Gandhi and P. Narasimhan.** s.l. : 26th IEEE International Conference on Distributed Computing Systems (ICDCS 2006), 2006. pp. 53-62.

42. *Securing the Deluge network programming system.* **P.K. Dutta, J.W. Hui, D.C. Chu and D.E. Culler.** s.l. : IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks, 2006. pp. 326-333.

43. *Secure code distribution in dynamically programmable wireless sensor networks.* **Mishra, J. Deng and R. Han and S.** s.l. : Proceedings of the fifth international conference on Information processing in sensor networks, 2006. pp. 292-300.

44. *TinyPK: securing sensor networks with public key technology.* **R.J. Watro, D. Kong, S.-F. Cuti, C. Gardiner, C. Lynn and P. Kruus.** s.l. : 2nd ACM Workshop on Security of ad hoc and Sensor Networks, Washington DC, 2004. pp. 59–64.

45. *Hash-based Secure Sensor Network Programming Method without Public Key Cryptography.* **Sokjoon Lee, Howon Kim, Kyoil Chung.** s.l. : Proc. the Workshop on World-Sensor-Web at International Conference on Embedded Networked Sensor Systems, 2006.

46. *Dynamic Program Update in Wireless Sensor Networks Using Orthogonality Principle.* **Das, M.L. Joshi, A.** s.l. : Communications Letters, IEEE, 2008. pp. 471-473.

47. *A ROM-friendly Secure Code Update Mechanism for WSNs Using a Stateful-verifier T -time Signature Scheme.* **Osman Ugus, Dirk Westhoff, and Jens-Matthias Bohli.** s.l. : Proceedings of the second ACM conference on Wireless network security, 2009. pp. 29-40.

48. *A Confidential and DoS-Resistant Multi-hop Code Dissemination Protocol for Wireless Sensor Networks.* **Hailun Tan, Diethelm Ostry, John Zic, Sanjay Jha.** s.l. : ACM, 2009.

49. **Wikipedia.** *http://en.wikipedia.org/wiki/Robustness.*

50. *http://en.wikipedia.org/wiki/Metadata.* **Wikipedia.**

51. *Seluge: Secure and DoS-Resistant Code Dissemination in Wireless Sensor Networks.* **Du, Sangwon Hyun Peng Ning An Liu Wenliang.** s.l. : IPSN 2008.

52. *An Efficient Scheme for User Authentication in Wireless Sensor Networks.* **Canming Jiang, Bao Li, Haixia Xu.** s.l. : 21st International Conference on Advanced Information Networking and Applications Workshops, 2007. pp. 438-442.

53. *Secure multi-hop network programming with multiple one-way key chains.* **Hailun Tan, Sanjay Jha, Diet Ostry, John Zic, Vijay Sivaraman.** s.l. : Proceedings of the first ACM conference on Wireless network security, 2008. pp. 183-193.

54. *Secure code distribution in wireless sensor Networks.* **Yee Wei LAW, James SING, Braden KIDD, Slaven MARUSIC.** s.l. : SENSEI, 2008.