TeleScribe: A Scalable, Resumable Wireless Reprogramming Approach

Min-Hua Chen
Department of Computer Science
National Tsing Hua University, Taiwan
orca.chen@gmail.com

Pai H. Chou
University of California, Irvine, CA USA and
National Tsing Hua University, Hsinchu, Taiwan
phchou@uci.edu

ABSTRACT

TeleScribe is a software mechanism for efficiently reprogramming embedded systems such as wireless sensor nodes over a shared communication link. One distinguishing feature is its ability to resume update on any node after power failure, link disconnection, or many other indefinite disruptive events. The nodes are guaranteed never to be left in a bad state as a result of such incomplete reprogramming procedures. Moreover, TeleScribe efficiently disseminates the binary image to as many nodes as possible, thereby minimizing redundant communication while ensuring that, in a later round as needed, all nodes receive packets that were lost earlier. Experimental results show TeleScribe to be the fastest and smallest among similar systems, achieving an update rate of about 95 bytes per node per second in a 100-node system. The total code size of our implementation on the node is around only 2 KB, making TeleScribe easily adaptable to a wide range of platforms with little overhead.

Categories and Subject Descriptors

C.3 [**Special-purpose and application-based systems**]: Real-time and embedded systems

General Terms

Design, Reliability

Keywords

Wireless sensor networks, resumable remote reprogramming, TDMA

1. INTRODUCTION

A robust and reliable mechanism to program sensor nodes is an important issue while building applications on wireless sensor networks (WSNs). Recent research works [1, 2] propose schemes to assist developers in building sensor applications and reducing the development cost. However, they assume that the sensor nodes have no hardware failures such as power failures and memory errors. Although detailed statics of such failure rates are not widely available, the users manuals of many modern digital devices warn the user about power interruptions during firmware update rendering the systems inoperable. To address this problem, this paper describes a remote reprogramming scheme that not only quickly reprograms all sensor nodes in the network but also enables all sensor nodes to recover from power failure during reprogramming. Therefore, our scheme greatly improves robustness over existing works.

Our work specially considers those applications of WSNs in wearable motion tracking and vital sign monitoring for health care, factory machinery monitoring, and laboratory experiments, where robustness is essential. Sensor nodes in the applications above are densely deployed on the order of hundreds to thousands of nodes in a relatively small area. We assume that the sensor nodes to be compact, simple and low cost, and that dense deployment with nontrivial data rates can be achieved with star topology and tiered networking. However, this work is equally applicable to reprogramming distributed, ad hoc WSNs where the sensor nodes have more resources than assumed in our experiments.

1.1 Problem statement

The goal of our work is to design an efficient, robust, scalable binary code reprogramming mechanism for WSNs. We assume that every sensor node in the network has a unique ID, and the base station always knows the maximum node ID in the network, which is an upper bound on the number of sensor nodes in the network. We also assume the network topology to be a star, where every node is in one-hop range from the base station. The final assumption is that all nodes in the network run the same binary image, although this assumption can be relaxed by adding a simple filter. In the upper tier, multiple base stations form a network of their own, and each base station may operate on a distinct frequency channel. Nodes may be dynamically associated with different base stations by a fast-handoff protocol [3].

1.2 Requirements

Remote reprogramming in wireless sensor networks is a multifaceted problem, which covers reliable data transmission, energy saving, data propagation protocol, hardware error detection, and satisfying constraints on sensor nodes, including the size of usable RAM and the limited capability of the radio transceiver. We first list the requirements of a remote binary code reprogramming scheme as follows:

1. Reliable Data Transmission

Remote reprogramming requires reliable delivery. Binary code must be transmitted completely and correctly; otherwise, the program will not work properly. All wireless links have inherently unpredictable packet loss. Either the sender or the receiver should keep track of the missing packets, and the sender or its delegate should re-send them to the receivers.

EMSOFT'10, October 24–29, 2010, Scottsdale, Arizona, USA.

2. Robust to Temporary Failure

Additional challenges are posed by temporary failures, particularly power outage. Battery-powered nodes may run out of power before they are replaced manually. Even nodes that harvest energy may temporarily run out of power before more energy becomes available again. Moreover, a worse problem is that the base station itself or a sender node may crash, thereby interrupting reprogramming of all of the nodes under its jurisdiction. In all cases, the remote reprogramming mechanism is expected to be able to resume the interrupted update process after the failed systems recover again.

3. Scalability

WSNs have grown in size quickly in recent years. A sensor network may contain several hundred to several thousand nodes. A remote reprogramming scheme should be able to distribute the binary data to the entire network quickly with minimal redundancy.

4. Compatibility

The remote reprogramming mechanism should be compatible with the original user program and the updated user program. The user program should not need to be aware of the reprogramming mechanism. With good compatibility, programmers should be able to build their own program with few restrictions on resource access.

5. Energy Consumption and Completion Time

Energy is the main limitation on many wireless sensor nodes. Some high energy-cost operations such as radio communication and external memory read/write [4] should be used with care. The remote reprogramming mechanism should avoid wasting energy. Completion time is another important metric. Remote reprogramming should complete the process within reasonable time and should not grow much with the size of the network.

6. Code Optimization

Wireless sensor nodes have limited storage space. Both the remote reprogramming program and the user program share the same storage space. Reducing the code size of the remote reprogramming program can increase the available storage space for user programs. Therefore, code size optimization of the remote reprogramming program is important on these wireless sensor nodes.

1.3 Objectives

Besides correctness of remote reprogramming, TeleScribe strives for robustness, quick completion time, high data rate, and small code size. Robustness is achieved by recording all necessary metadata during the update process in the non-volatile memory on the nodes, and it enables each node to resume the update process after interruptions such as power failure. Code written to nonvolatile memory is verified for correctness. Quick completion time is achieved by pre-scheduled broadcast communication with a novel group-based NACK protocol for retransmitting lost packets. The code size of our implementation is about 2.7KB as a stand-alone program. For compatibility, the user program only has to follow some guidelines as imposed by our current platform such as the interrupt service routine (ISR) sharing and the code allocation at compile time. With better hardware and software support, these restrictions can be lifted. Basically the user program is not aware of the remote reprogramming program. Users do not have to combine another boot loader with their program [5] to keep the reprogramming mechanism working.

The rest of this paper is organized as follows. In Section 2, we discuss related works in the remote reprogramming area. Section 3 describes our design and implementation in detail. We evaluate TeleScribe and present the experimental results in Section 4. Section 5 concludes and discusses directions for future research.

2. RELATED WORK

In this section, we discuss related work on remote reprogramming. They can be divided into reliable transmission, data dissemination protocols, and remote reprogramming mechanisms.

2.1 Reliable Transmission

Reliable transmission is essential in the remote programming area. A binary program must be received and written completely and correctly, or else it cannot work as expected.

RMST [6] partitions large data objects into small pieces of code called *segments* and transmits them separately instead of sending the whole large data objects at once. The receivers reassemble small segments after receiving them. A receiver sends negative acknowledgments (NACKs) to its one-hop neighbors to request lost segments. The hop-by-hop error recovery reduces the retransmission times.

PSFQ [7] (pump slowly, fetch quickly) suggests sensor nodes to send data at a relatively slow speed and to fetch lost data at a high speed. In this way, a sensor node can fetch lost data from its one-hop neighbors when the lost data is still in the neighbors' local cache, so that it does not have to fetch the lost packet from the source, which may be a few hops away.

Our work, named TeleScribe, borrows ideas such as fragmentation/reassemble, NACKs, and hop-by-hop error recovery from previous work. The fragmentation/reassemble approach can help sensor nodes send large binary data, and hop-by-hop error recovery reduces the cost compared to end-to-end error recovery.

2.2 Dissemination Protocol

Different data dissemination protocols make different assumptions about the network. In the classic flooding protocol, a node unconditionally forwards data to all its neighbors. This simple protocol causes the *broadcast storm* problem [8], where redundancy and collision impair the efficiency and reliability. Another variant of flooding is called *gossiping*, which sends messages to a randomly chosen neighbor. In this way, only one copy of a given message exists in the network, and thus gossiping reduces the data traffic and saves energy. However, gossiping distributes data slowly.

SPIN [9] suggests a three-stage handshaking protocol using three types of messages: advertisement (ADV), request for data (REQ), and data message (DATA). When a node has data to share, it advertises ADV messages to its neighbors. A node receives an ADV message and replies with a REQ message if it wishes to receive the data. A node sends a DATA message if it receives a request for that data. SPIN provides a family of protocols to solve the *implosion* problem and the *overhear* problem under different assumptions. TeleScribe uses an enhanced version of the three-stage handshaking protocol with SPIN.

2.3 Remote Reprogramming

For remote reprogramming, Multi-hop Over-the-Air Programming (MOAP) [10], Deluge [2], and Multi-hop Network Reprogramming (MNP) [4] partition the binary program into segments of data and transmit the segments separately instead of sending the whole binary image directly. The receiver maintains a bitmap to detect the packet loss. If a packet loss is detected, then the receiver sends a NACK message to the sender to request the lost packet. This NACK-based retransmission approach can reduce the control traffic. The fragmentation/reassemble approach also improves the propagation performance. Deluge and MNP pipeline the transfer of segments to achieve full capabilities of the network.

Considering the data dissemination protocols, the naive flooding approach causes the broadcast storm problem, and several so-

lutions have been proposed to address this problem. Trickle, Deluge, and MNP take an *epidemic/gossip* approach to propagating data [11, 12]. Trickle uses a "polite gossip" policy to suppress network traffic, where a node broadcasts an advertisement message to its neighbors but remains quiet if it has recently heard an identical advertisement message. TinyOS supports remote programming by XNP [5]. It is a single-hop reprogramming solution with an expensive loss detection mechanism. Moreover, programmers must combine another boot loader with their program to reserve the remote reprogramming function.

Difference-based approaches have been proposed to reduce the cost of remote reprogramming [13]. Instead of sending the whole binary program, a difference-based technique sends the difference between two different versions of a binary program when sending the difference part is cheaper. Maté [14] is a stack-based virtual machine, where a virtual program is represented as one or more *code capsules*. The virtual program is propagated by sending small code capsules; thus the cost of sending virtual programs is lower than sending binary programs. However, the virtual machine approach is more restricted, because sometimes the virtual machine itself may need to be reprogrammed.

TeleScribe broadcasts control messages and data to all the nodes. TeleScribe shares ideas with [6, 4, 2, 10], where every receiver maintains its bitmap to track lost packets. The loss detection method is cheaper than XNP, which scans the EEPROM to detect lost data. To solve the collision problem, TeleScribe takes a time division multiple access (TDMA) approach to sending NACKs to the source node. The one-to-many data broadcast and TDMA improve the performance of TeleScribe.

2.4 Power Interruption During Remote Programming

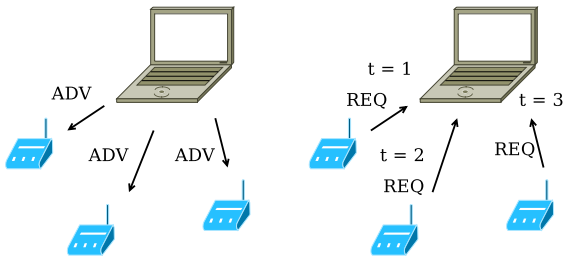
If remote reprogramming is interrupted due to a power failure or system crash, whether on the nodes or the base stations, previous reprogramming approaches might not be able to recover from the interrupted reprogramming process, and the sensor node may not work again because of the incomplete binary program. TeleScribe considers the power failure problem during reprogramming process. The status of reprogramming is reserved in the non-volatile memory, and the update process can be recovered after the power becomes available again. This design improves the robustness, not just reliability of remote programming.

3. DESIGN AND IMPLEMENTATION

This section describes the design and implementation details of TeleScribe. The two main components in our implementation are the sensor nodes and the base station. We first show our protocol design and explain our implementation.

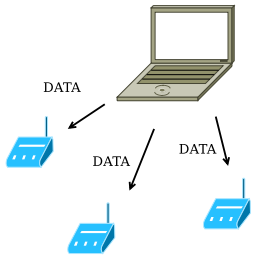
3.1 Protocol Description

Figure 1 shows an example scenario, where the base station has a new binary code image to propagate to the entire network. It first broadcasts an advertisement message (ADV) to all the nodes to notify them of the newly available program. The ADV message contains several attributes, the most important of which is the version number of the new binary program. If the new version number differs from its own number, then the sensor node understands that it is a new program and initializes the remote code update process. After receiving the ADV message, every sensor node sends its request (REQ) to the base station. To avoid packet collision, every sensor node is assigned a unique time slot to send its request. Finally, the base station broadcasts the requested data (DATA) to the



(a) Base station broadcasts an advertisement message.

(b) Nodes reply requests to the base station.



(c) Base station broadcasts data.

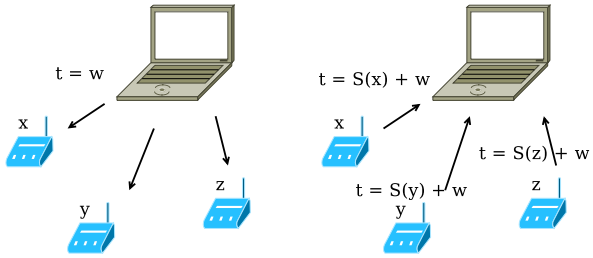
Figure 1: Example scenario.

sensor nodes. This process is repeated until the base station receives no more REQ messages.

It is entirely possible that some nodes still might not have been updated completely by the time no node sends more REQ messages. This is because REQs are effectively NACKs, which confirm not-receiving, but the absence of NACKs is not sufficient to conclude receiving, even though in practice all our nodes are updated successfully in the laboratory experiments. In order to eliminate the uncertainty of the update status of every sensor node, we make a slight modification to the three-stage handshaking protocol: all completely updated sensor nodes still send REQ messages after receiving an ADV message from the base station, but the REQ messages request nothing. This effectively tells the base station that the node has been updated successfully. When all nodes have been updated completely, the base station broadcasts a start signal to the entire network to notify the sensor node to start running the newly updated program. If incompletely updated nodes still exist after no node NACKs, either due to power outage or other temporary failure, once these nodes recover from failure, they will be ready to resume the update process without being stuck in a bad program state.

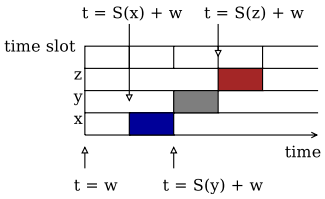
3.1.1 TDMA for REQ

In order to apply the TDMA technique to the REQs from the nodes, we implement a simple time synchronization mechanism. According to our assumption, every member node has a unique ID, and the node ID also determines the time slot of each node. We also assume that the base station knows the maximum node ID in the network, so that the base station can adjust its TDMA frame to cover all time slots. After receiving an ADV message, a node waits until its time slot comes and sends the REQ message to the base station. We also assume that all member nodes are in the radio range of the base station, and thus the transmission delay



(a) Base station broadcasts an advertisement message.

(b) Nodes reply requests to the base station.



(c) Time slots allocation.

Figure 2: The TDMA technique.

is negligible. Figure 2 illustrates our TDMA technique for REQ scheduling, where three nodes have unique node IDs x, y, and z. The unique IDs are mapped to their unique time slot by the time slot function S.

$$S(t) = t \cdot C \tag{1}$$

where the constant C is the length of a single time slot. After some experiments, we set C to 3.75 ms for our specific radio and microcontroller speed.

3.1.2 *Improved TDMA for REQ by Progressive Doubling of Groups*

An obvious disadvantage of the fixed TDMA technique is that the communication time is proportional to the number of nodes in the network. When the size of the network is large, the cost of communication time will be considerable if not prohibitive. The following describes an improved TDMA scheme. The base station first broadcasts special ADV messages to the entire network. The ADV messages have a special attribute called the *group number*. A node is a *group member* if its node ID is *less than or equal* to the group number. Only group members are asked to send their REQ messages to the base station after receiving the ADV message with the group number. Non-members may not send REQ messages but may keep listening to the radio channel and grab any data that they might not have received. When the base station broadcasts DATA messages in response to the REQ messages from the group members, the non-members can receive them, too.

With this group concept, our approach is to progressively increase the group size until all nodes are covered. That is, starting from a relatively small group number, after a whole round without effective REQ messages from the member nodes, the base station doubles its group number (thereby doubling the frame size) and sends another ADV message again. Since the new group is a superset of the old group, members of the old group can still send REQ messages to the base station for its lost data. The doubling action is repeated until the group number is equal to the size of

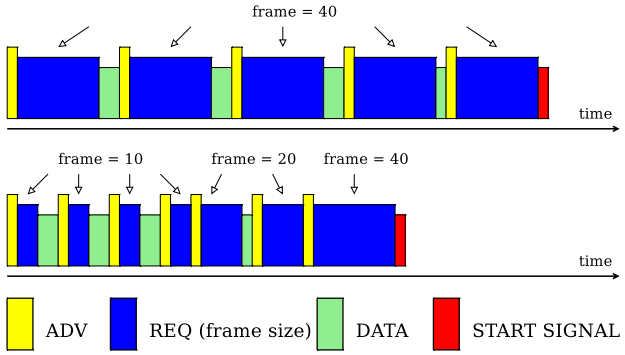


Figure 3: (a) TDMA with fixed frame size (b) Improved TDMA technique with progressive group doubling.

the entire network. In this way, the base station can reprogram the entire network by the REQ messages from many different small groups instead of receiving the REQ messages from the entire network every time. This turns out to be a very effective technique for speeding up the TDMA phase. We have implemented both TDMA techniques and evaluate them in our experiments.

The difference between the fixed and improved TDMA techniques is explained in Figure 3. Figure 3(a) shows the time usage of the fixed TDMA technique. To simplify explanation, we assume that there are 40 sensor nodes in the network, and each of them has a 1-unit-long time slot. As the figure shows, the fixed frame size is 40 and sum of the TDMA frames for receiving REQ messages from sensor nodes dominates the reprogramming time.

Figure 3(b) illustrates the time allocation of the improved technique with the initial group size of 10 (so is the frame size). In the first three rounds of handshaking, the base station handshakes with a 10-member group and broadcasts the DATA messages requested by the group. All sensor nodes can grab the DATA messages if they have not received them. In the fourth handshaking round, the base station received no effective REQ messages and doubles the group number. In the fifth round, the base station receives a small number of effective REQ messages and hence broadcasts a small number of DATA messages. After the sixth round (group size of 20) of no REQ, in the seventh round, the frame size is increased to 40 and can cover all time slots. After this round, the base station receives no effective messages and broadcasts a start signal to the network. Comparing both techniques in Figure 3, we can see significant time improvement contributed by the improved TDMA technique.

3.1.3 *Error Recovery*

To increase reliability, every member sensor node keeps a bitmap to track packet loss, and the bitmap is initialized to zero. After receiving an ADV message, a node sends a REQ message to the base station. The REQ packet contains the node ID and its bitmap, so that the base station knows the reprogramming progress of every sensor node after gathering REQ messages. Finally, the base station re-broadcasts the lost data to the network.

3.2 The Sensor Nodes

Before describing the implementation details of the sensor nodes, it is important to understand the address space usage on the sensor nodes when writing a new binary program to the nonvolatile memory. In our design, the address space is divided into two areas: the system area and the user program area. The *system area* stores the essential program with the remote reprogramming function, named the "image loader," and the *user program* area stores user programs.

```
struct image_struct {
        unsigned char status;   // image status
        unsigned char version;  // version number
        unsigned int  id;       // node ID
        unsigned char count;    // segment counter
        __code char *bm_ptr;    // bitmap pointer
}
```

Figure 4: The `image_struct` data structure in C language.

By default, the system area is loaded in the sensor nodes and never overwritten by other user programs, but the user programs can be replaced by other user programs.

The readers may wonder if the system area can be reprogrammed by the remote reprogramming mechanism. The answer is positive, but there are some risks. First, since the reprogramming information is recorded in the non-volatile memory, reprogramming the system area destroys this information. Hence, the reprogramming cannot be recovered if there is any interruption during the reprogramming. Second, if the successfully updated program has some serious bugs with the remote reprogramming functions, then there may be no way to reprogram the buggy program by using the buggy program.

To explain the details of remote reprogramming and reliable transmission, one important data structure must be introduced: a data structure named `image_struct`. It stores important information such as the status of the user program, the version number of the current user program, the node ID, and other information, as Figure 4 shows. The `image_struct` is stored in the non-volatile memory of the sensor nodes and is always up-to-date. The non-volatile property makes remote reprogramming more reliable, because the image loader can resume reprogramming even after a power interruption has occurred during remote reprogramming, all by using this information.

3.2.1 Remote Reprogramming Mechanism

Now we explain our design of TeleScribe. When the system boots up, the image loader first checks the `image_struct` for the status of the user program. If the user program is ready to run, then the image loader will jump to the starting address of the user program; otherwise, the image loader assumes the current user program is incomplete and enters remote reprogramming mode to resume updating the user program.

An important question is, when should the image loader change the status of the user program from ready to remote reprogramming mode? The answer is upon receiving an ADV message during execution of the user program. An ISR for the transceiver is shared by both the image loader and the user program. When the base station broadcasts an ADV message to a sensor node, the ISR parses the ADV message and knows that a new program is available, and the ISR changes the status of the user program from ready to initial mode. In this implementation we assume single-threaded code without race conditions; in a more general execution model, additional mechanisms for atomicity may be required. In initial mode, the image loader initializes a bitmap in its non-volatile memory to track packet loss and changes the status to remote reprogramming mode.

When the image loader is in remote reprogramming mode, it follows the three-stage handshaking protocol. The sensor node waits for an ADV message from the base station. After receiving the ADV message, the sensor node sends its ID and bitmap (REQ) back to the base station, and then waits for DATA messages. When a DATA message arrives, the image loader checks its bitmap to see whether the DATA has already been received. If the DATA message has not been received, then the image loader writes the segment of binary program to its non-volatile program memory and updates its bitmap. Otherwise, the image loader drops the DATA message. When the remote reprogramming is complete and the sensor node receives a start signal from the base station, the image loader changes the user program status to ready and reboots.

3.3 The Base Station

In TeleScribe, the base station connected to the host computer is the only source of a new user program, and the base station takes care of all requests from the entire network in a short time. Therefore, we build a base station using a 16-bit microcontroller unit (MCU) that is powerful compared to the 8-bit MCU on the highly resource-constrained nodes. In a more distributed version, nodes similar to the iMote with much more resource may be considered for performing the same tasks as our base station here.

3.3.1 Components of the Base Station

To build a base station with high processing speed and high data receive/transmit speed, we choose a PC to be the host to process the REQ messages from sensor nodes, and a base station communicate with the sensor nodes via a compatible transceiver module. We implemented our base station with an evaluation board and an RF transceiver module. The base station communicates with the PC host via Ethernet and with the sensor nodes via its RF transceiver.

3.3.2 Manipulating the Binary Program

To address the packet loss problem, the base station and the sensor nodes cooperate to increase the transmission reliability. The sensor nodes keep their bitmaps to detect packet loss, and the base station divides the user program into small pieces of code called *segments*. The segments of code are packetized and sent to the sensor nodes.

4. EVALUATION RESULTS

This section shows the evaluation results of TeleScribe. We first describe the hardware platforms, software tools, and the experimental environment. We evaluate completion time in both fixed TDMA and improved TDMA techniques. We also assess the effectiveness of the resumable feature of TeleScribe. To stress test our scheme, we repeat the remote reprogramming at least 2000 times.

4.1 Experimental Setup

Our experimental setup consists of the hardware platform and software development tools.

4.1.1 Sensor Platform

TeleScribe has been implemented on the Eco sensor node [15], an ultra-compact system that is $1cm^3$ in volume and weighs 2 grams. It consists of four subsystems: MCU/Radio, Sensors, Power, and the expansion port. The 8051-compatible MCU has a 512-byte ROM for the bootstrap loader, a 4K-byte RAM for data and code, a 4K-byte serial EEPROM for code and read-only data, and a 9-channel ADC. Eco has a built-in triaxial acceleration sensor ($\pm$3g) and a 16-pin expansion port. The transceiver on Eco is the Nordic nRF2401, which is integrated with the nRF24E1 MCU, and the transmission speed can be set to either 1 Mbps or 250 Kbps. TeleScribe is compact enough to fit in Eco's small memory.

4.1.2 Base Station

We also built a base station by connecting the Freescale DEMO-9S12NE64 evaluation board with the Nordic nRF24L01 transceiver

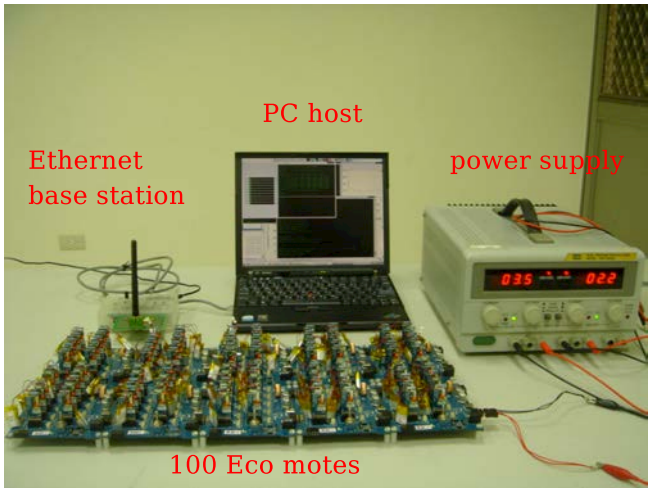


Figure 5: Experimental setup.

module. The board uses the Freescale MC9S12NE64 MCU with an integrated Fast Ethernet MAC/PHY controller. The Ethernet base station has a Fast Ethernet (100 Mbps) interface, enabling it to communicate with the PC host quickly.

4.1.3 *Software*

The binary code for the Eco nodes is built by the Small Device C Compiler (SDCC), version 2.6.0. The development tool for the Freescale evaluation board is CodeWarrior. The control program on the host PC is written in Python.

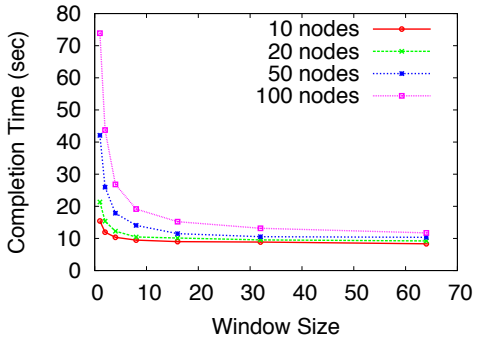
4.1.4 *System Setup*

Figure 5 shows our entire setup. All Eco sensor nodes are connected to the cascaded battery charging boards to be powered during the experiments. The power switch on the charger board allows us to turn off and on a group of up to 10 Eco nodes at a time instead of individually. The transceiver power level of every Eco sensor node is configured for 0 dBm, and the data rate is configured for 1 Mbps. All Eco sensor nodes are placed within the radio range of the base station. The PC host and the base station are connected via an Ethernet cable.

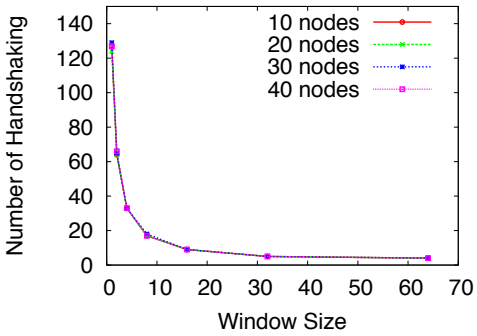
4.2 Results: Completion Time

This subsection presents the experimental results and analyzes the performance of TeleScribe in terms of the completion time. The *completion time* here is defined as the time from the start of the remote reprogramming to the successful completion of reprogramming all sensor nodes in the network. Note that we mean *fully acknowledged* completion, i.e., not only have all nodes been programmed successfully but the host has also received the positive confirmation from all nodes. Other papers may report completion times that are *visually* confirmed by the authors, but their protocols might not have a way for the nodes to acknowledge their successful programming to the host.

We repeat reprogramming the entire network by alternating between two different user programs. The size of each user program is 998 bytes and is partitioned into 63 segments of 16 bytes each. The size of the user program is limited by the size of the user program area. Every sensor node is assigned a unique node ID, and the IDs are sequential, starting from 1. In other words, if there are n sensor nodes in the network, the IDs are $1, 2, ..., n$. During our



(a) Completion time.



(b) Number of rounds of handshaking.

Figure 6: Experimental results of completion time and number of rounds of handshaking for Fixed TDMA.

experiments, the remote reprogramming is repeated at least 2000 times, and it also shows the reliability of TeleScribe.

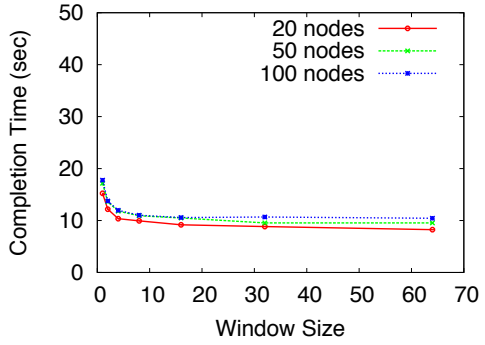
We test window sizes of $1, 2, 4, ..., 64$ (i.e., powers of 2) with different network sizes and measure the completion time of remote reprogramming. The *window size* in our experiment is the number of data segments that the base station broadcasts to the entire network at a time.

4.2.1 *Fixed TDMA*

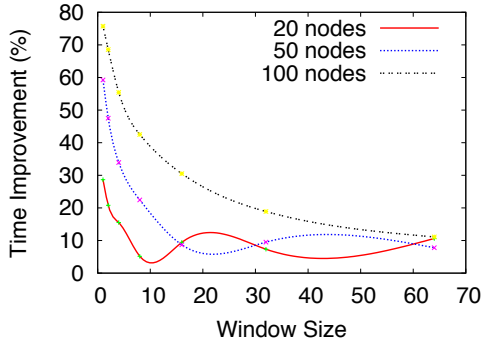
Figure 6(a) shows the relationship between the window size and the completion time for the fixed TDMA scheme. The completion time decreases rapidly as the window size increases from 1 to 8 and decreases slowly as the window size increases from 16 to 64. The larger speed-up from 1 to 8 is due to the fact that the number of rounds of ADV-REQ-DATA handshaking decreases as the window size increases.

For example, if the size of a user program is 32 segments and the window size is 8, then the number of rounds of handshakings is 4 in the best case. The smaller speed-up from 16 to 64 is due to the small size of the user program. The window sizes are large enough to complete the small size reprogramming in a few broadcasts. Although a larger window size can complete the reprogramming in fewer steps, after recovering the lost packets, the transmission time is not very different from that of smaller window sizes.

Figure 6(b) shows the number of rounds of handshaking in this experiment. An interesting fact is that the number of rounds of handshaking is almost the same over different window sizes. It implies that the data transmission time does not cause the difference in completion time of different network sizes. Combining Figures 6(a) and 6(b), we can conclude that the size of the network domi-



(a) Complection time.



(b) Completion time improvement(%).

Figure 7: Completion time of improved TDMA technique with initial group number 10.

nates the transmission time, since the larger network has a longer TDMA frame to communicate with all sensor nodes.

4.2.2 *Improved TDMA*

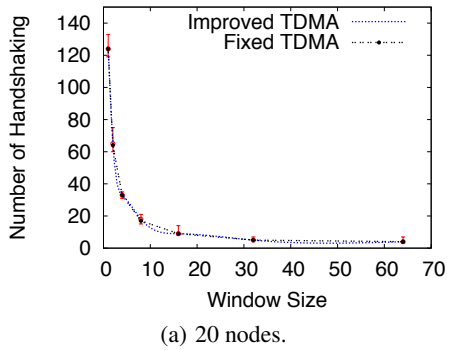
In Section 3.1.2, we propose an improved TDMA technique to reduce the time spent by the fixed TDMA technique. Figures 6(a) and 7(a) show the completion time of the fixed and improved TDMA techniques. The initial value of the group number is 10. The results show significant improvements in larger network sizes. The improvement is not obvious when at large window sizes, because the cost of fixed TDMA handshaking and the cost of the improved TDMA are almost the same.

Figure 8 shows that, despite incurring slightly more handshaking, the improved TDMA technique completes reprogramming significantly faster than the fixed TDMA technique. This is because the handshaking cost increases progressively and is much cheaper for smaller group sizes in the improved TDMA scheme. To understand this fact, we express the main components of the completion time as:

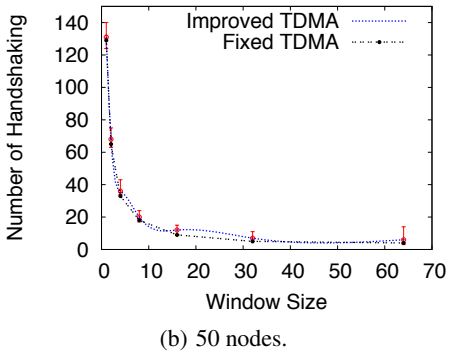
$$\text{Time}_{\text{completion}} \cong \sum_{1 \le k \le n} F_k + \sum_{1 \le k \le n} T_k \qquad (2)$$

where n is the number of rounds of handshaking, and F_k and T_k are the frame size and data transmission time at the k^{th} round.

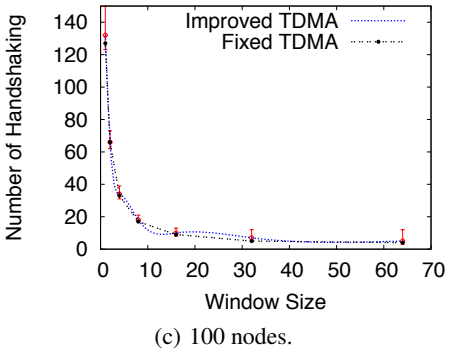
In the fixed TDMA technique, all the frame sizes remain the same during the remote reprogramming and are long enough to cover all time slots. On the other hand, the improved TDMA technique (Section 3.1.2) first selects a small group of sensor nodes and handshakes with them. This way, it needs only a small frame size to cover this group. When the base station receives no effective REQ messages from the original group, the group size and the



(a) 20 nodes.



(b) 50 nodes.



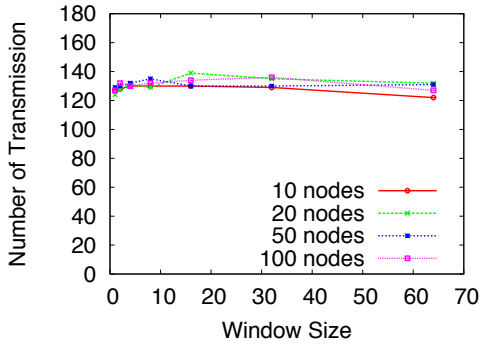
(c) 100 nodes.

Figure 8: Number of rounds of handshaking of improved TDMA techniques with group number 10.

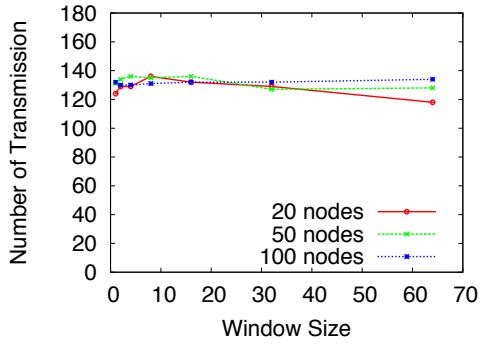
frame size are progressively doubled until the group size is equal to the network size. The difference in the total number of transmissions between the two techniques is not obvious, as Figure 9 shows. There, the completion is dominated by the sum of frame sizes in each handshake. In a large network, the fixed TDMA technique has a long and fixed frame size, but the improved TDMA technique has mostly shorter frame sizes and very few longer frame sizes.

Back to Figure 8, the small amount of increment means that the DATA messages requested by the group members are greatly beneficial to the non-members. Therefore, the non-members have fewer or no lost packets to recover. To support this conjecture, we examine the experimental data and count the number of transmissions at different group numbers. After analyzing the data, we find a consistent result that most of the transmissions occur in the initial small group in all different window sizes. Figure 10 shows the results when the window size is 64. We find that most of the transmissions occur when the group number is 10, and there are much fewer transmissions in other larger group sizes.

Figure 11 show the overall throughput of different network sizes,



(a) Fixed TDMA.



(b) Improved TDMA.

Figure 9: Number of transmissions of fixed and improved TDMA.

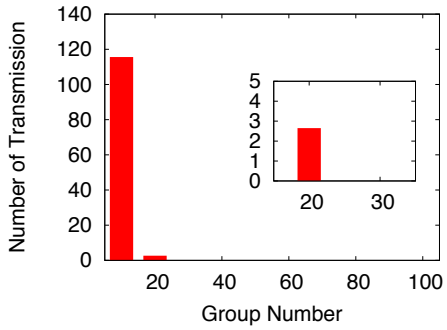
using our improved TDMA scheme. The overall throughput is defined to be

$$\text{Overall Throughput} = \frac{\text{Size(user program)}}{\text{Completion time}} \quad (3)$$

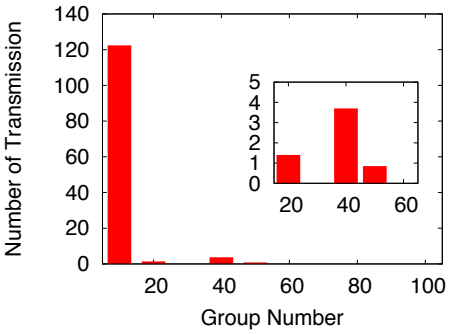
Combining Equation (3) and the definition of the completion time, which is the time from the start of the remote reprogramming to the successful completion of reprogramming all sensor nodes in the network, the overall throughput can be considered to be the lower bound of the throughput of all the nodes in the network. In other words, the throughput of every sensor node is higher than or equal to the overall throughput. The throughput decreases as the network size grows. This decrease is due to the fact that the TDMA techniques need larger frame sizes in larger networks, regardless of fixed or our improved TDMA scheme. The overall throughput of TeleScribe is about 95 bytes per second per node in a 100-node network, and this is considered relatively fast. To recall, our completion time is what the host reports after full confirmation with every node, not just when each node finishes programming itself but might not have told the host. This also means in reality, our nodes finish programming much earlier. The high throughput shows that the improved TDMA technique can significantly reduce the overhead of fixed TDMA technique and thus makes TeleScribe an efficient remote reprogramming mechanism. Table 1 lists the throughput of existing works.

4.3 Discussion: TDMA or CSMA

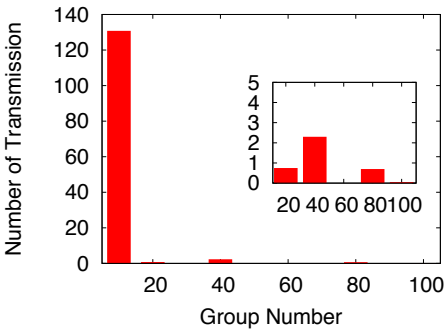
Some readers may speculate that the completion time will become shorter if we choose TDMA protocol instead of CSMA protocol for REQs. This seems plausible, since when the group size is doubled, the TDMA frame time will cover nearly half of the



(a) 20 nodes.



(b) 50 nodes.



(c) 100 nodes.

Figure 10: Number of transmission at different group number, window size = 64.

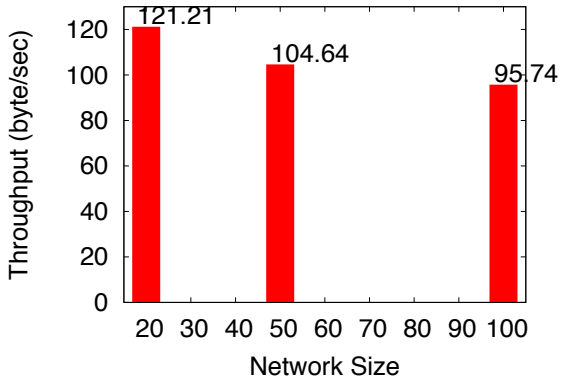


Figure 11: Overall throughput in different network sizes(Improved TDMA).

Table 1: Overall throughput comparison.

Work	Network Size	Throughput (bytes/s)	Platform
TeleScribe	100	95	Eco mote
Deluge	75	90	Mica2
MNP	25	70.4	Mica2

Table 2: TDMA time frame usage.

Network size	stage 1 (s)	stage 2 (s)	stage 3 (s)	total (s)
20	0.20	0.06	0.08	0.34
50	0.17	0.55	0.19	0.91
100	0.17	0.75	0.38	1.30

completely reprogrammed sensor nodes. In other words, half of the doubled frame time will be wasted. Figure 10 shows more details. After the complete reprogramming of the initial group, only a small number of sensor nodes need retransmission. If we switch to the CSMA protocol, it seems we may have a good chance to save the doubled frame time.

To estimate the benefits of the CSMA protocol, we analyze the experimental data of the TDMA approach and divide the frame time of the reprogramming process into three stages: reprogramming of the initial group, reprogramming of the other sensor nodes, and the final confirmation by all the sensor nodes. Table 2 shows the total frame time consumed in the three stages, using the TDMA protocol. Switching to the CSMA protocol may reduce the time of stage 2. However, stages 1 and 3 would require the entire network to send REQs to the base station, and CSMA would cause serious collision problems during these two stages.

Table 3 shows the best-case completion times for CSMA. It turns out CSMA performs better than TDMA during stage 2 for network sizes of 50 and 100 but worse for the network size of 20. However, for stages 1 and 3, CSMA performs exponentially worse with the size of the network. The major reason is that all the sensor nodes have to compete for a limited period of time to send their REQs. When the time is shorter, the CSMA protocol can save more time in stage 2. On the other hand, a shorter time will cause more collisions in stages 1 and 3. Hence, the CSMA protocol might not perform as well as expected.

4.4 Result: Resumable Reprogramming

To assess the effectiveness of our resumable reprogramming feature, we injected power failures by turning off the power at various times during the remote reprogramming. In all of our tests, all the interrupted sensor nodes can fully resume the remote reprogramming process after the power is turned on again [16]. In fact, the resumable reprogramming feature helps a great deal during our experiments, not necessarily due to node failure, but due to the base station overheating and crashing, which caused interruptions to the remote reprogramming for the entire network. The resumable reprogramming feature enables the sensor nodes to resume update despite incomplete binary images on all these nodes and saves much time.

Table 3: Estimated CSMA time frame usage.

Network size	stage 1 (s)	stage 2 (s)	stage 3 (s)	total (s)
20	0.23	0.11	0.23	0.57
50	0.50	0.23	0.45	1.18
100	1.13	0.38	1.32	2.83

5. CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

We propose TeleScribe, a scalable and resumable remote reprogramming approach for WSNs. TeleScribe is capable of recovering from bad states caused by interruptions such as power failures during the remote reprogramming, thereby enhancing the robustness of remote reprogramming. In order to achieve high-speed reprogramming, TeleScribe uses a Fast-Ethernet base station and uses a progressive frame-size doubling TDMA technique for lost-packet requests. Evaluation results show an overall, *fully-verified* throughput of 95 bytes/sec in a 100-node network. It means each node effectively receives *at least*, not *at most*, 95 bytes/s, since the completion time of the network is determined by last the node that completes the reprogramming and positive confirmation with each node.

TeleScribe make three contributions in the remote reprogramming area. First, it provides a large scale, high-speed, reliable remote reprogramming mechanism for WSNs. Second, TeleScribe builds a resumable reprogramming feature so that a node with an incomplete user program can recover from the bad state and eventually receive a complete user program. Finally, the code size of TeleScribe is about 2K bytes. The small code size can easily fit the platforms with small memory spaces such as Eco sensor nodes.

5.2 Future Work

TeleScribe can be extended in several directions, including more general network features, further improving the TDMA, and power management.

5.2.1 Larger Scale WSNs

Results presented in this paper are based on one base station handling at least 100 to 200 nodes with little difficulty. To scale to larger networks, it will be necessary to consider either multiple base stations or multi-hop topologies. Multiple base stations that employ frequency-division multiple access (FDMA) with fast handoff [3] already enables TeleScribe to scale up by two orders of magnitude with no increase in reprogramming time in a two-tier network. This translates into 10000 to 20000 nodes and 100 to 200 base stations, as limited by the number of available frequency channels, even if their coverage areas completely overlap. Whether by peers or base stations, it will be important to consider reprogramming authentication and program data validation for security reasons. Even without malicious attacks, nodes may contain stale versions of a program and should be suppressed from further propagation. The current scheme for program version number generation based on MD5 cannot tell which version is newer, since the version number is more like a hash key, although this can be easily extended with more bits. Upon receiving code segments, the nodes should verify the program data and catch inconsistency either intentionally injected by attackers or introduced by bit errors in memory buffer.

5.2.2 Further Improved TDMA Technique

The improved TDMA technique (Section 3.1.2) significantly reduces the overhead of the fixed TDMA technique (Section 3.1.1); however, the doubled frame size still covers nearly half of completely reprogrammed sensors, causing wasted time. To eliminate the wasted time slots, the base station can broadcast the reprogramming status of each sensor node to the entire network. Every sensor

node can recalculate its new time slot after receiving the information, thereby reducing the waste of time slots.

5.2.3 Power Management

Energy is a main constraint in WSNs. TeleScribe assumes all sensor nodes keep listening to the radio for the messages from the base station. The idle listening undoubtedly wastes a considerable amount of energy. Thus, reducing the idle listening time is an important issue for future work. We already considered several ways to address the energy waste problem. Sensor nodes can adopt new radio modules with lower power consumption to save energy. Another solution is to schedule sensor nodes to turn on the radio only when necessary. To achieve the scheduling solution, there must be a precise time synchronization mechanism. Except turning off the radio when it is idle, the energy consumption can be controlled by dynamically adjusting the power level of the transceiver. The sensor node can monitor its link quality and lower its power level if the quality is acceptable or increases its power level when the quality is going down. The power level should be minimized if the link quality is acceptable. Such power management schemes are currently being implemented in the handoff mechanism [3].

Acknowledgments

This work was sponsored in part by the National Science Foundation CAREER Grant CNS-0448668, CNS-0721926, the National Science Council (Taiwan) Grant NSC 96-2218-E-007-009, and Ministry of Economy (Taiwan) Grant 96-EC-17-A-04-S1-044. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

6. REFERENCES

[1] Adam Dunkels, Niclas Finne, Joakim Eriksson, and Thiemo Voigt. Run-time linking for reprogramming wireless sensor networks. In *SenSys'06*, pages 1–3, November 2006.

[2] Jonathan W. Hui and David Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *SenSys'04*, pages 3–5, November 2004.

[3] Chung-Yi Ke, Nai-Yuan Ko, Chih-Hsiang Hsueh, Chih-Hsuan Lee, and Pai H. Chou. EcoPlex: Empowering compact wireless sensor platforms via roaming and interoperability support. In *Proceedings of the Sixth Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services (MobiQuitous 2009)*, Toronto, Canada, July 13-16 2009.

[4] Sandeep S. Kulkarni and Kimin Wang. MNP: Multihop network reprogramming service for sensor networks. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICSCS'05)*, 2005.

[5] Crossbow Technology Inc. *Mote in-network programming user reference*. `http://www.tinyos.net/tinyos-1.x/doc/Xnp/pdf`.

[6] Fred Stann and John Heidemann. RMST: Reliable data transport in sensor networks. In *IEEE Workshop on Sensor Net Protocols and Applications (SNPA)*, May 2003.

[7] Chieh-Yih Wan, Andrew T. Campbell, and Lakshman Krishnamurthy. PSFQ: A reliable transport protocol for wireless sensor networks. In *WSNA'02*, September 2002.

[8] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu. The broadcast storm problem in a mobile ad hoc network. In *International Conference on Mobile Computing and Networking*, 1999.

[9] Hoanna Kulik, Wendi Heinzelman, and Hari Balakrishman. Negotiation-based protocols for disseminating information in wireless sensor networks. *Wireless Networks*, 2002.

[10] Thanos Stathopoulos, John Heidemann, and Deborah Estrin. A remote code update mechanism for wireless sensor networks. Technical report, CENS Technical Report, 2003.

[11] J. Pereira, L. Rodrigues, M.J. Monteiro, R. Oliveira, and A.-M. Kermarrec. NEEM: Network-friendly epidemic multicast. In *Proc. 22nd Symposium on Reliable Distributed Systems (SRDS)*, pages 15–24, October 2003.

[12] Indranil Gupta, Anne-Marie Kermarrec, and Ayalvadi J. Ganesh. Efficient and adaptive epidemic-style protocols for reliable and scalable multicast. In *Proc. IEEE Symposium on Reliable Distributed Systems (SRDS)*, 2002.

[13] Niels Reijers and Koen Langendoen. Efficient code distribution in wireless sensor networks. In *WSNA*, September 2003.

[14] Philip Levis and David Culler. Maté: A tiny virtual machine for sensor networks. In *ASPLOS*, 2002.

[15] Chulsung Park and Pai H. Chou. Eco: Ultra-wearable and expandable wireless sensor platform. In *Proc. Third International Workshop on Body Sensor Networks (BSN 2006)*, pages 162–165, April 2006.

[16] Min-Hua Chen. TeleScribe – 100 nodes with power failure. `http://www.youtube.com/watch?v=bu90xtj-yfY`.

[17] Fred Stann, John Heidemann, Rajesh Shroff, and Muhammad Zaki Murtaza. RBP: Robust broadcast propagation in wireless networks. In *Sensys'06*, pages 1–3, November 2006.

[18] David Chu, Lucia Popa, Arsalan Tavakoli, Joseph M. Hellerstein, Phillip Levis, and Scott Shenker. The design and implementation of a declarative sensor network. In *SenSys'07*, pages 6–9, November 2007.

[19] Philip Levis, Neil Patel, David Culler, and Scott Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. Technical report, University of California, Berkeley, 2004.

[20] S.T. Hedetniemi S.M. Hedetniemi and A.L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18:319–349, 1988.

[21] Chulsung Park and Pai H. Chou. AmbiMax: Automous energy harversing platform for multi-supply wireless sensor nodes. In *IEEE SECON 2006 proceedings*, 2006.

[22] Farhan Simjee and Pai H. Chou. Everlast: Long-life, supercapacitor-operated wireless sensor node. In *ISLPED 2006 proceedings*, 2006.

[23] Chung-Ye Ke. EcoMAC: A fast-handoff, collision-free, lightweight MAC protocol for multi-channel heterogeneous wireless sensor networks. Master's thesis, National Tsing Hua University, 2008.

[24] David Wikie. *EEPROM Endurance Tutorial*, 2005.

[25] Microchip Technology Inc. *Everything a System Engineer Needs to Know About Serial EEPROM Endurance*, 1992.