2013-07-15

Dynamic Near Field Communication Pairing For Wireless Sensor Networks

Steven Charles Cook
Brigham Young University - Provo

Dynamic Near Field Communication Pairing for Wireless Sensor Networks

Steven C. Cook

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

J. Kelly Flanagan, Chair
Dan R. Olsen Jr.
David W. Embley

Department of Computer Science

Brigham Young University

July 2013

ABSTRACT

Dynamic Near Field Communication Pairing
for Wireless Sensor Networks

Steven C. Cook
Department of Computer Science, BYU
Master of Science

Wireless sensor network (WSN) nodes communicate securely using pre-installed cryptographic keys. Although key pre-installation makes nodes less expensive, the technical process of installing keys prevents average users from deploying and controlling their own WSNs. Wireless pairing enables users to set up WSNs without pre-installing keys, but current pairing techniques introduce numerous concerns regarding security, hardware expense, and usability.

This thesis introduces dynamic Near Field Communication (NFC) pairing, a new pairing technique designed for WSNs. This pairing overcomes the limitations of both key pre-installation and current pairing techniques. Dynamic NFC pairing is as secure as using pre-installed keys, requires only inexpensive NFC hardware, and is easy to use since the user simply holds nodes close together to add them to a network.

A sample application shows the power of dynamic NFC pairing. The user adds sensors and actuators to a WSN by holding each node close to a central node or network coordinator. Data readings stream instantly from each sensor to a web page where the user may view data as well as click buttons to cause events to occur on the actuators. This happens quickly and securely without exposing the user to the complexity of cryptographic keys.

Keywords: near field communication, wireless pairing, security, wireless sensor networks

ACKNOWLEDGMENTS

I thank Dr. Kelly Flanagan for mentoring me despite his various obligations and responsibilities. He helped transform our serendipitous encounter into an invaluable learning experience.

I also thank my wife, Erin, for her continual support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Imagine living in a fully-automated home. The heater, air conditioner, and windows adjust automatically depending on who is inside. The doors lock, the blinds close, and the dishwasher pauses when you are in the shower. The sprinklers schedule themselves depending on recent rainfall and the weather forecast. While at work you check to see if the curling iron was left on and turn it off using your cell phone. All of these scenarios and countless more are possible today using wireless sensor network technology.

1.1 Wireless Sensor Networks

A *wireless sensor network* (WSN) is comprised of several wireless nodes that are physically distributed throughout an environment. Nodes gather and report information about their surroundings (e.g., temperature, motion detection, etc...), typically through a single gateway node. In addition to reporting data to a user, the gateway node may also allow the user to control the network nodes by sending them individual commands (e.g., lock the door).

WSN nodes communicate using wireless standards such as ZigBee or Z-Wave which have been specialized for sensor networks. These standards allow nodes to sleep most of the time, waking only to report data and receive commands. Therefore, WSN nodes are referred to as *lightweight wireless nodes* because they require only simple hardware and consume little power when compared to nodes running WiFi or other wireless standards. Since a single WSN may include hundreds of nodes, a good lightweight wireless protocol helps reduce the expense of the overall WSN by simplifying individual nodes.

A typical lightweight wireless node is composed of a small micro-controller (with access to flash memory and RAM), radio, power supply, and some kind of sensor and/or actuator. Sensors gather information about their environment; actuators cause events to occur. Lightweight wireless nodes are commonly powered with batteries. Due to the power-efficient nature of lightweight wireless standards, a sensor node that sleeps and periodically reports data, often referred to as a *sleepy sensor*, may last for years on a single small battery. Actuator nodes which execute user commands, such as opening and closing window blinds, normally draw power from wall outlets and remain awake continuously. This allows for instant responsiveness to commands.

1.2 Importance of WSNs

WSN applications are rapidly emerging in home security, smart entertainment, industrial control, building automation, medical data collection, surveillance, precision agriculture, vehicular traffic management, and disaster detection [8, 9]. WSNs, when connected to the Internet, provide users with instant access to environments from remote locations. WSNs can both make automatic decisions, or simply provide information so the user may make informed decisions. For example, a security system might automatically lock the doors when it detects an intruder; an agricultural WSN might provide a farmer with water and soil measurements so he may choose how to best fertilize the land.

Wireless sensor networks are quickly gaining popularity. ABI Research predicts that in the year 2016, the IEEE 802.15.4[1] integrated circuit market will ship over 850 million WSN chipsets [1]. As sensor technology and lightweight wireless protocols continue to advance, WSNs may revolutionize the way users monitor and interact with their environments.

[1]IEEE 802.15.4 is a standard for the physical layer and media access control for low-rate wireless personal area networks. It serves as a foundation for various WSN specifications.

1.3 Problem to be Solved

Although WSN technology is capable of a variety of practical applications, WSNs are not yet popular in the home. This is due to three factors that make creating such networks difficult for average homeowners: security, usability, and expense.

First, wireless communication must be secure. Without any security mechanism, wireless communication may be attacked in a variety of ways. Two wireless nodes may overcome these attacks by using cryptographic keys. The keys must be negotiated in a safe manner (i.e., not via insecure radio waves). Currently, WSN nodes are pre-installed with keys by the manufacturer or by a specialized technician. This makes it difficult for homeowners to set up secure WSNs without help. Because keys are compiled into the WSN nodes' firmware, a homeowner would need to access the firmware source code (which is usually proprietary) for a node, type in the keys, re-compile the firmware for the target hardware, and then upload the firmware executable to the node using its bootloader (e.g., over-the-air, dongle, etc...). The user must then repeat this process for every node that will join the WSN.

The inability to control one's WSN reveals a second issue: usability. The process of setting up and controlling a WSN must be simple (i.e., usable) for the user, but users are currently unable to pre-install keys without direct help. One solution to this problem is for nodes to negotiate keys through pairing rather than by key pre-installation. *Wireless pairing* is the process of dynamically negotiating a key between two nodes without having agreed upon any settings or secrets beforehand. In order to pair securely, both nodes need access to an out-of-band (OOB) channel which is unavailable to attackers. The nodes then use the OOB channel to share a key and begin secure communication.

The third obstacle preventing WSNs from becoming popular in the home is keeping expenses low so users purchase wireless nodes. Although pairing might allow average homeowners to easily and securely set up WSNs, secure pairing is always more costly than key pre-installation because it requires additional hardware in order to access an OOB channel.

This thesis presents a new pairing method that overcomes the challenges of security and usability while minimizing the necessary expenses. This new solution is suitable for lightweight wireless nodes. Users will be able to set up WSNs and automate their homes in new and creative ways without having any expertise regarding wireless technology. Tech-savvy hobbyists will no longer be the only ones preheating the oven while driving home for dinner.

Chapter 2

Pairing Techniques and Their Limitations

Research proposes two primary solutions to overcome the challenges of security, usability, and expense in setting up WSNs: improve communication strategies that currently use key pre-installation, or abandon key pre-installation entirely and use pairing instead. Researchers in favor of key pre-installation provide new techniques for deriving and distributing safer keys [6, 13, 15, 20, 21]. Although the use of better keys strengthens network security and keeps node expenses low, it does not solve the usability problem; lightweight nodes must be configured offline and given carefully-selected key material by hand [10]. Unfortunately, lightweight wireless standards such as ZigBee are currently only able to communicate securely if nodes are pre-installed with keys [3].

Other researchers propose pairing, as opposed to key pre-installation, as a user-friendly way to associate two wireless nodes. These proposals may be categorized as user-assisted pairing, hardware-assisted pairing, and in-band pairing. We now evaluate these three pairing categories in terms of the following criteria:

- **Security**: Is the pairing method secure? If pairing is not secure, it fails to qualify as a secure pairing method by definition.
- **Hardware Cost**: What additional hardware is needed to implement the method? If a pairing method requires special hardware that is expensive, then the method is not acceptable.
- **Usability (User-Action)**: Does the method require user-action? If a pairing method requires user-action, how much must the user do? Is the action difficult or error-prone?

2.1 User-Assisted Pairing

User-assisted pairing methods require the help of a user to complete the pairing. The user helps by either authenticating a key or by transferring an entire key from one device to the other. The type of user-action is categorized as follows:

- **Output Comparison**: Bluetooth's Secure Simple Pairing (SSP) - Numeric method [4], Wireless USB Association (WUSB) - Numeric [4], Manual Authentication (MANA) II [11], and Short Authenticated Strings (SAS) [14, 29] are all pairing techniques that involve visual comparisons. In each of these techniques, both pairing devices display a number or string calculated from the pairing process. If the user confirms that both devices show the same output then the pairing was secure.
- **Typing**: Bluetooth's SSP - Passkey Entry method [4], Wi-Fi Protected Setup (WPS) - In-Band [4], MANA I [19], and MANA III [18] each require the user to either read a number from one device and type it into the other, or to type the same string into both devices.
- **Button Presses**: Button-Enabled Device Association (BEDA) - Button Timing and BEDA - Blinking Light [27] are techniques that require the user to press buttons. With button timing, the user presses and holds buttons on each device simultaneously. The duration of the button-press becomes a shared secret which is used to derive a key. In blinking light, the user transfers a key by pressing a button on one device whenever the other device's light blinks.

Table 2.1 shows a summary of each user-assisted pairing category. Although comparing output, typing, and pressing buttons may seem simple, each introduces opportunities for user errors. All three also require additional hardware. Adding keyboards or screens to lightweight wireless nodes may be impractical. Adding buttons may be an acceptable solution because buttons are inexpensive.

Pairing Method	Security	Hardware Cost per Device	User-Action
Output Comparison	Secure	Display and Button	Visual Comparison
Typing	Secure	Keyboard or Display	Typing
Button Presses	Secure	Button	Button-presses

Table 2.1: Summary of User-Assisted Pairing Methods

2.2 Hardware-Assisted Pairing

Hardware-assisted pairing methods allow devices to pair using an out-of-band (OOB) channel that an attacker is not likely to have access to. Devices are able to agree on a key using the OOB channel without any help from the user because the channel is safe. Once both devices have a key, they may associate using radio transmissions encrypted with the key, resulting in a secure pairing. In other words, two devices simply need access to a common OOB channel in order to make pairing secure. Several pairing methods take advantage of the following four OOB channels:

- **Near Field Communication**: WPS and SSP both have OOB procedures that use the Near Field Communication (NFC) channel to negotiate keys [4]. The two devices must be close together for NFC to work. These methods will be explored in greater depth later.
- **Camera**: Seeing-Is-Believing (SiB) - Barcode [23] and SiB - Blinking Light [5] both use a camera on one device to read a key from the other by looking at either a light or a barcode. The camera on one device must point towards the light or barcode on the second device.
- **Infrared**: This technique transfers a key between two devices using infrared light as the OOB channel [31]. The infrared transmitter on one device must point towards the receiver on the other device.
- **Ultrasonic**: Ultrasonic Distance-Bounding is a technique that uses ultrasonic sound as the OOB channel to derive a key [7].

Table 2.2 shows a summary of each hardware-assisted pairing category. Although the user may need to orient devices in order for hardware-assisted pairing methods to work properly, there is no user-action involved in the actual key transfer. Each method requires specialized hardware (e.g., a camera) to function. Extra hardware can be devastatingly costly for lightweight nodes.

Pairing Method	Security	Hardware Cost per Device	User-Action
NFC	Secure	NFC reader/writer or tag	None
Camera	Secure	Camera or Light/Barcode	None
Infrared	Secure	Infrared Transmitter/Receiver	None
Ultrasonic	Secure	Ultrasonic Transmitter/Receiver	None

Table 2.2: Summary of Hardware-Assisted Pairing Methods

2.3 In-band Pairing

SSP - Just Works and WPS - Push Button [4] both perform a key exchange using each device's regular radio. Once established, the keys are simply not authenticated. Although these methods require no extra hardware to operate and require no user-action, they are not secure [16]. It may seem surprising that these insecure pairing methods are commercially popular. Perhaps users prefer a simple pairing method at the expense of security, or perhaps they do not realize that these methods are insecure.

2.4 Pitfalls of Pairing

Research agrees that user-assisted pairing methods always result in some degree of user-error [24, 26, 28, 30]. Additionally, each of these pairings takes several seconds to complete. The combination of errors and pairing duration may frustrate the user.

Li et al. [17] suggest that user-assisted pairing may be suitable for WSNs if using group device pairing. This allows a user to pair multiple nodes simultaneously using a single action, thus significantly reducing the user's involvement.

Gollakota et al. [25] propose a pairing method that operates entirely over the regular radio medium in such a way that the keys do not require authentication to be considered safe. The only action required of the user is to press a button on both devices in a manner similar to WPS - Push Button pairing. This pairing method was implemented on WiFi nodes and consumes an extreme amount of power until the pairing is complete. Future research may refine this method until it is suitable for lightweight nodes.

2.5 Pairing for WSNs

Although the ideal pairing method is not physically possible, we define an *ideal pairing* method to be any pairing method that is secure, requires no user-action, and adds no additional hardware cost to wireless nodes. We have evaluated several pairing methods categorized as user-assisted, hardware-assisted, and in-band pairing. Of course, none of the pairing methods meet the definition of an ideal pairing method as shown in Table 2.3.

Pairing Method Category	Secure	Extra Hardware	User-Action
User-Assisted	Yes	Yes	Yes
Hardware-Assisted	Yes	Yes	No
In-Band	No	No	No
Ideal	Yes	No	No

Table 2.3: Pairing Methods vs. Ideal Pairing

User-assisted pairing is secure, but it requires both additional hardware and user-action. Although comparing output, typing, or pressing buttons may seem simple, we know from previous work that each action introduces user-error and requires time. User-assisted pairing also requires additional hardware that may be expensive for lightweight nodes.

Hardware-assisted pairing does not depend on the user for key authentication, thus making NFC, camera, infrared, and ultrasonic pairing methods user-friendly. However, like user-assisted pairing, each method requires specialized hardware to function.

In-band pairing requires no extra hardware to operate and needs no user-action, but it is not secure. Recent research methods providing safe in-band pairing algorithms consume too much power.

We conclude that there is currently no secure pairing method for lightweight wireless nodes that is secure, inexpensive, and usable. We also conclude that hardware-assisted pairing methods most closely resemble the ideal method. Therefore, we propose a new pairing method that is hardware-assisted in which the required hardware is as inexpensive as possible.

Chapter 3

Proposed Solution

3.1 Thesis Statement

A new pairing method for lightweight wireless nodes is made possible through the proper use of Near Field Communication (NFC) dynamic tags; this pairing is secure, inexpensive, and usable.

3.2 Near Field Communication

Of the hardware-assisted pairing methods that we previously evaluated, the NFC methods are of particular interest. NFC operates on the 13.56 MHz radio band and has a maximum read distance of only a few centimeters. In order to communicate, NFC requires one device to be NFC-enabled, meaning that it is an NFC reader/writer. The other device may also be an NFC reader/writer, or it may be an NFC tag. NFC reader/writers are more expensive, in the range of $25-$50; NFC tags are inexpensive typically ranging from 20 cents to a dollar.

Current NFC pairing methods allow nodes to negotiate secure keys entirely using NFC. When two nodes are brought close together, their NFC hardware begins to communicate. If both nodes have an NFC reader/writer, the nodes agree on a key which is then used to encrypt future radio transmissions. Because NFC has such a short range, an attacking device is unable to detect the NFC signals and intercept the key unless it is within centimeters of the two nodes at the moment of pairing. We assume that a user would notice foreign hardware in such close proximity and refuse to pair.

3.3 Limitations of NFC

Simple Secure Pairing (SSP) and Wi-Fi Protected Setup (WPS) are two methods that currently allow for NFC to provide the OOB channel for pairing. Although both are usable, each pairing method has its own weakness. We now explore the limitations of SSP - OOB due to hardware expense and the security issues of using WPS - OOB.

3.3.1 NFC Expense

In SSP - OOB, each device is required to have an NFC reader/writer in order to pair. This is a secure and usable solution, but not cost-effective; in the context of a wireless network, every node must have an expensive NFC reader/writer.

3.3.2 NFC Security

In WPS - OOB, NFC is used as the OOB channel to transfer a key from an NFC tag on one node to an NFC reader/writer on another. Unfortunately, this requires one node to be pre-configured with a key. Since NFC tags are built into stickers that are stuck onto a node's packaging, that node's processor has no way of getting data to or from its own NFC tag. Therefore, the NFC tag must be pre-installed with a key, and the node's non-volatile memory must also be pre-installed with the same key. Then, when a different node reads the key out of the NFC tag, both nodes have the same key.

This situation presents a few difficulties. First, someone must pre-install the nodes that have NFC tags with security keys. Again, the average user is probably not going to be able to do this because it requires access to the firmware source code. If a technician or manufacturer were to pre-install the NFC tag nodes, they might as well have simply pre-configured the whole network with keys. The nodes would no longer need NFC hardware at all and we would be back to the original problem of how to avoid key pre-installation.

If the user asks for assistance from the manufacturer, the manufacturer could easily pre-install the security keys. However, since a node does not know when its tag is changed,

tags must be written by the manufacturer permanently. That way the data persists and both a node and its tag can be programmed with matching data. This is a security weakness because the data on that tag is available for an attacker to read from the moment it gets printed. An attacker may be able to read your device's NFC tag before you even purchase it. Likewise, after a sensor is already joined to a network, an attacker could come and read the key using NFC because the key always remains in the tag.

3.4 Dynamic NFC Pairing

We now present a new pairing method based on NFC that is secure, inexpensive, and usable. We term this method *dynamic NFC pairing*.

Dynamic NFC pairing makes use of a recently-invented technology called NFC dynamic tags to overcome the difficulties of using NFC for wireless pairing. NFC dynamic tags are similar to NFC tags except that they are accessible to a micro-controller via wire. In other words, a wireless node may read the data in its NFC tag through a direct connection.

In dynamic NFC pairing, one node (node A) has an NFC reader/writer, and the other (node B) has an NFC dynamic tag. Neither node is pre-configured with a key. To initiate pairing, the user holds the two nodes close together. Once within NFC range, the reader/writer on node A selects a key and writes it to the dynamic tag on node B. The dynamic tag then passes the key to node B's micro-controller. Node B's micro-controller then clears and turns off its dynamic tag, thus making it impossible for attackers to later read the key using an NFC reader/writer. At this point, both nodes have the same key and may begin to communicate securely via radio.

Dynamic NFC pairing, when used in a WSN, comes very close to the ideal pairing goal. It is secure because there are no pre-installed keys in the firmware for an attacker to probe and because NFC works only over such a short distance. It is usable because the user is not directly involved in the key transfer. The user only needs to hold two nodes close together and the hardware takes care of the rest. Lastly, the necessary hardware for dynamic NFC

pairing will likely become inexpensive in the near future. In a WSN, only one node, such as the gateway node, needs an NFC reader/writer. All other nodes may have inexpensive NFC dynamic tags. This means that the average additional cost per node for pairing approaches the cost of an NFC dynamic tag as more nodes are added to a network. The exact prices of NFC dynamic tags and NFC reader/writers depend on a variety of factors that will be explained in Section 5.3.3.

Other advantages of dynamic NFC pairing include always joining the intended network and doing so quickly. Users will know which network they are joining; they will always join the network belonging to the node physically close to their device. Pairing will appear to be almost instantaneous to the user.

Chapter 4

Implementation

This chapter describes an implementation of a WSN using our dynamic NFC pairing technique. From the user's perspective, a central wireless node acts as the network coordinator. To add a sensor or actuator to the network, the user simply flips the node's power switch to the "on" position and holds it within centimeters of the coordinator. Moments later the sensor or actuator is added securely to the network and begins reporting data. The user may now place the node anywhere within range of the WSN and the node continues to work.

In this implementation, the user may open a web page associated with the network. All of the sensor and actuator data is listed and updated in real-time. Buttons on the web page allow the user to control the network and perform immediate actions such as turning switches on and off. The whole process is meant to be intuitive and simple for non-technical users.

4.1 System Architecture

An overview of the system architecture is shown in Figure 4.1. At its core, this system is a WSN built on the ZigBee wireless standard. A single ZigBee coordinator node is the center of the WSN and both collects data and sends commands to the attached nodes. The coordinator is also an Internet gateway allowing it to connect to a web server. This allows the user to browse to a web page and view sensor data and send commands to the actuators.

The major components of our WSN nodes are shown in Figure 4.2. Figure 4.2a shows that in our implementation, a sensor or actuator node consists of the EM357 system on a

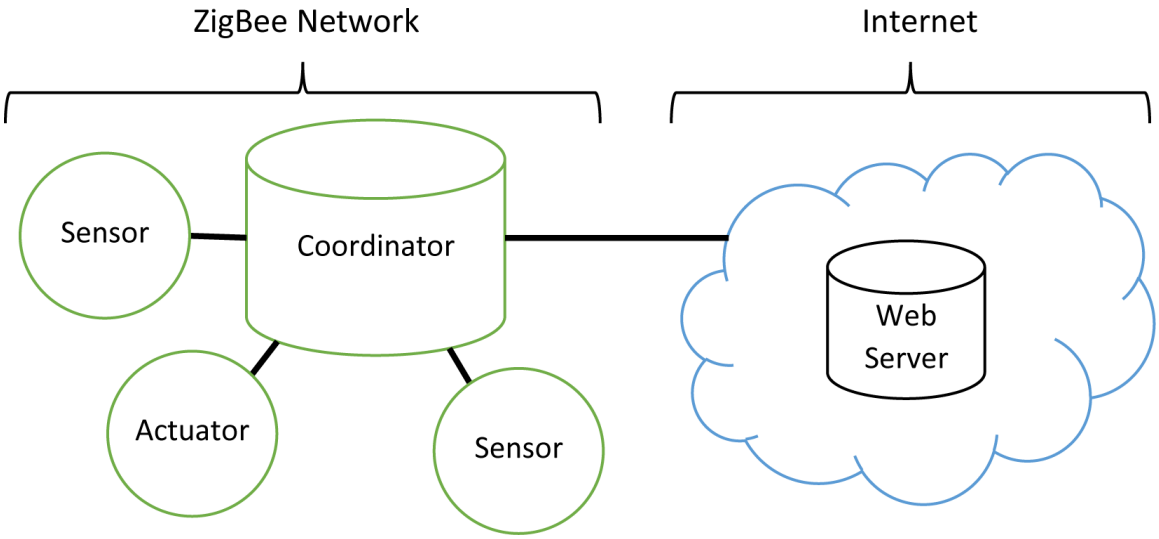
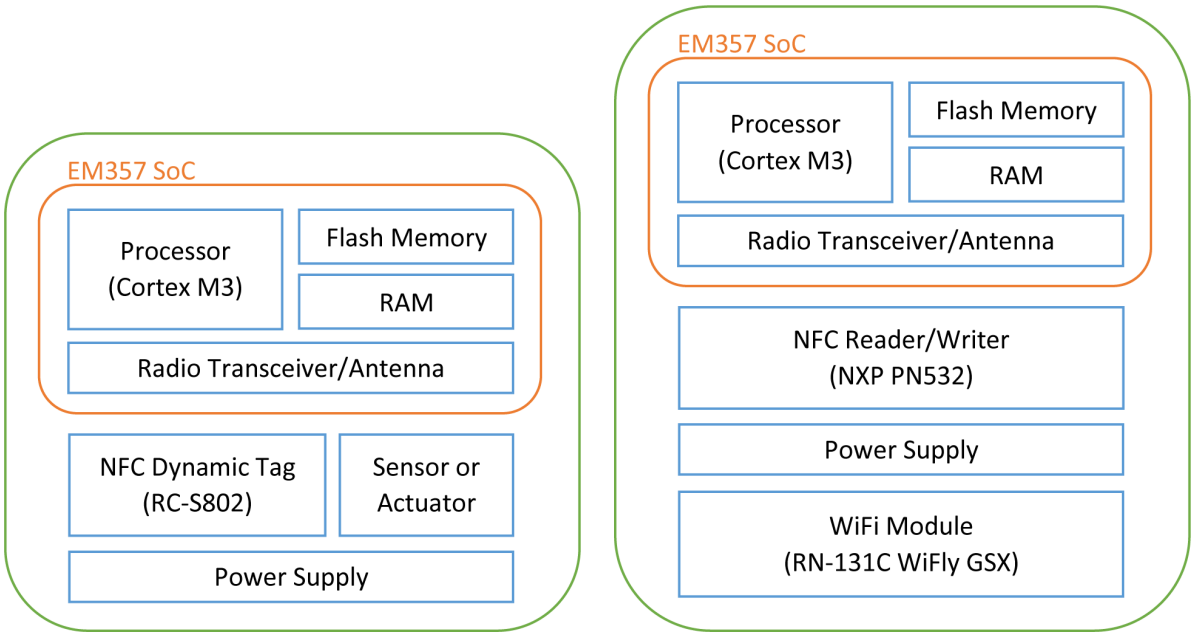


Figure 4.1: Overview of System

The system is composed of two networks: a ZigBee WSN (in the home) and the Internet (in the cloud). The ZigBee WSN consists of a single coordinator node and multiple sensor and actuator nodes. The coordinator communicates with the sensors and actuators in the WSN and with a web server in the Internet.



(a) Sensor or Actuator Node (b) Coordinator Node

Figure 4.2: Hardware Components of WSN Nodes

chip (SoC), an NFC dynamic tag, a sensor or actuator, and a power supply. The coordinator node is different in that it has an NFC reader/writer instead of an NFC dynamic tag, and it has a WiFi module instead of a sensor or actuator as shown in Figure 4.2b.

We now explain the main components of the system in greater detail including the NFC pairing, EM357 SoC and ZigBee protocol, application firmware, WiFi module, and web server.

4.2 Dynamic NFC Pairing

The purpose of this system is to demonstrate dynamic NFC pairing. This pairing requires the WSN coordinator to have an integrated NFC reader/writer and for sensor and actuator nodes to each have an integrated NFC dynamic tag. The coordinator is in charge of allowing sensors and actuators to join the WSN. When the user wants to add a sensor or actuator node to the network, he holds the node close to the coordinator. The coordinator sends the joining node the network key via NFC. The sensor or actuator is then able to join the network securely using the key.

4.2.1 Working Reference

Our task was to integrate an NFC reader/writer into the coordinator and NFC dynamic tags into sensor and actuator nodes. To be efficient, we first bought a commercial NFC reader/writer, the Sony Contactless IC Card Reader/Writer (RC-S330), and some compatible FeliCa NFC tags. This hardware was not part of the final WSN nodes, but rather acted as a working reference with which we could both observe proper functionality and test our own NFC hardware for correctness. In other words, we could later test our NFC dynamic tag with the commercial NFC reader/writer, and we could test our coordinator's NFC reader/writer with a commercial NFC tag.

To get the commercial contactless IC card reader working, we needed to use its Application Programming Interface (API) which was designed to interface with the user

Figure 4.3: NFC Working Reference

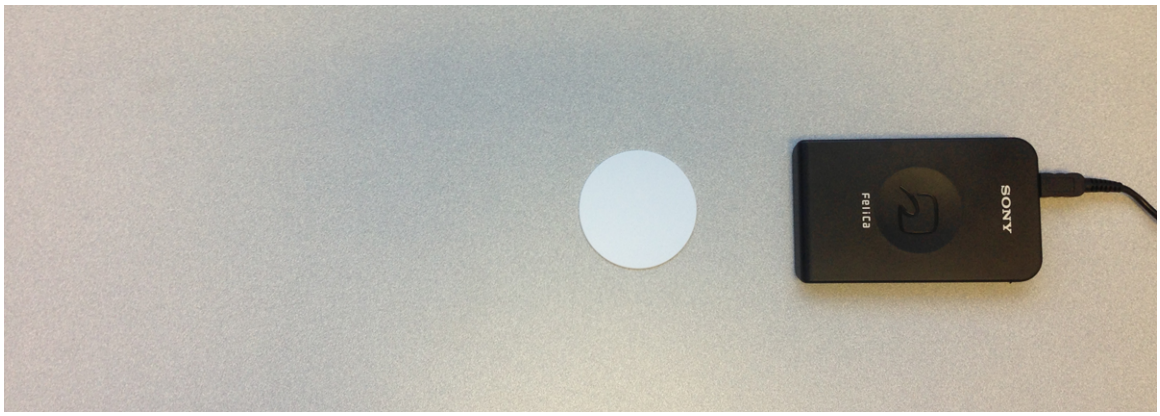
The commercial NFC reader/writer (right) and commercial NFC tag (left) served as a reference point for our NFC hardware. When the tag is moved close to the NFC reader/writer, the NFC reader/writer is able to read and write data to the tag.

through a web browser. This required us to program the card reader with MXML and ActionScript using Adobe Flex. We were able to successfully produce a web page that used the card reader's API to read and write to FeliCa tags. We were able to use this simple application as a working reference so that we could develop and test our own NFC reader/writer and dynamic tag individually. Figure 4.3 shows the working NFC hardware.

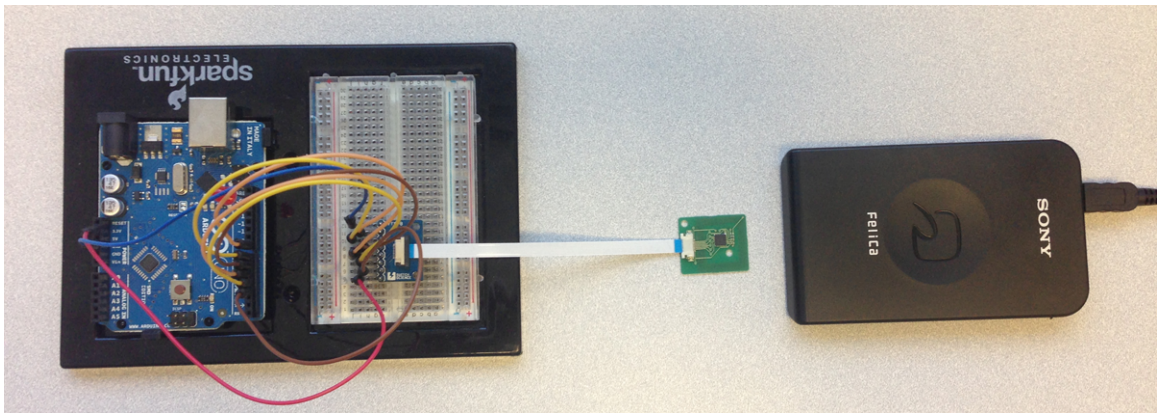
4.2.2 New NFC Hardware

The next step in creating our system was to prepare an NFC reader/writer and a dynamic tag that could be integrated into WSN nodes. We selected Sony's RC-S802, also known as the NFC Dynamic Tag "FeliCa Plug," as the dynamic tag hardware because it was the only dynamic NFC tag available on the market at the time. We selected an open-source Arduino NFC shield for the NFC reader/writer. The shield includes the NXP PN532 IC as well as built-in NFC coils. We used the open source Arduino platform as a familiar environment for rapid prototyping.

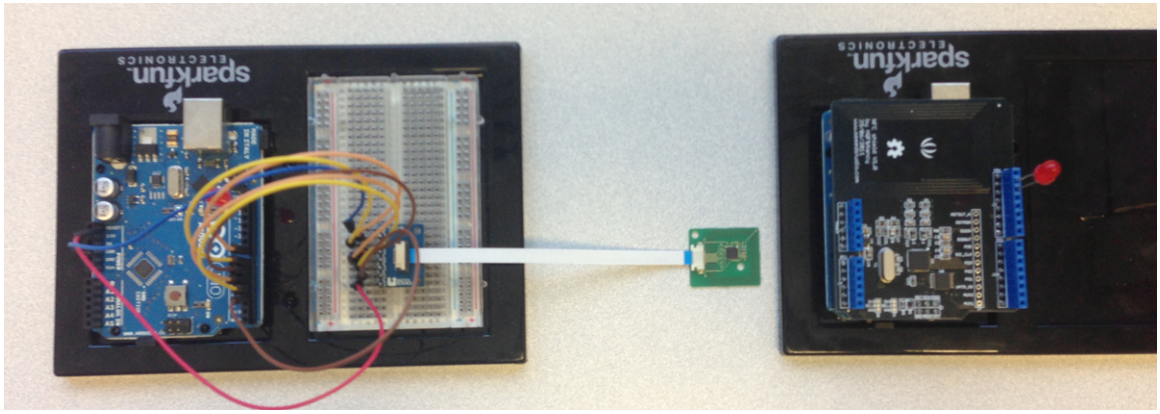
The contactless IC card reader that we set up earlier helped us correctly program the dynamic tag. Figure 4.4 shows our transition from using the card reader and NFC tag to



(a) Left: FeliCa NFC tag. Right: IC Card Reader/Writer and desktop computer.



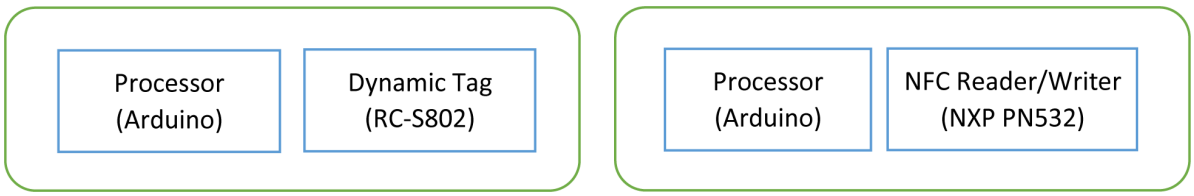
(b) Left: NFC dynamic tag and Arduino. Right: IC Card Reader/Writer and desktop computer.



(c) Left: NFC dynamic tag and Arduino. Right: NFC reader/writer (shield) and Arduino.

Figure 4.4: Transition From IC Card Reader and Tag to Arduino

In (a), the NFC tag is not connected to any external processor, so we replace the tag with an NFC dynamic tag in (b). The NFC dynamic tag connects to a prototyping processor, an Arduino, allowing us to control the tag and read data out of it. The NFC reader/writer in (a) and (b) is connected to a desktop computer's processor. We replace the reader/writer with an open-source reader/writer in (c) that can be connected to an Arduino. We later replace both Arduinos with WSN node processors.



(a) Sensor or Actuator Node (b) Coordinator Node

Figure 4.5: Nodes with NFC Hardware

using the open-source shield and NFC dynamic tag. We first wired the dynamic tag to an Arduino and wrote custom software to drive it. We knew the dynamic tag worked correctly when it could be used in place of the FeliCa tags in the contactless IC card reader browser application. The next step was to program a second Arduino to drive the NFC shield. Once the shield worked with the FeliCa tags, we tested the NFC shield with the dynamic tag. We wrote small programs for both Arduinos so that the shield and tag could exchange security keys and other information. This demonstrated our ability to use NFC as the OOB channel for secure pairing.

At this point, the WSN nodes had only a processor and NFC hardware as shown in Figure 4.5. The nodes could do nothing besides exchange a key via NFC.

4.3 Ember Radio

Now that our nodes were able communicate using NFC, we needed to add radio transceivers and a wireless network protocol so that nodes could form a WSN. The Arduino processor was good for a prototype because it is able to control the NFC hardware very similarly to how a WSN node processor would. However, because Arduino does not include a radio transceiver, we traded the Arduino for the more powerful EM357.

4.3.1 EM357

The Ember (now Silicon Labs) EM357 is a high-performance ZigBee SoC. The package includes the 32-bit ARM® Cortex™-M3 processor, 192 kB of Flash memory, 12 kB of

RAM, and a 2.4 GHz IEEE 802.15.4 radio transceiver. Most importantly, the EM357 runs EmberZNet PRO, a very prominent ZigBee-compliant wireless stack in the market. The accompanying development kit comes with a rich set of tools to help in configuring, creating, and debugging wireless ZigBee applications.

4.3.1.1 Introduction to ZigBee

ZigBee [3] is a lightweight wireless protocol based on the IEEE 802.15.4 standard. It was designed specifically for low-power and low-cost sensor and control networks. The ZigBee standard is maintained by a group of companies called the ZigBee Alliance.

A ZigBee network has one central node called a network *coordinator*. The coordinator generally manages all the security keys for the network and acts as a data sink. Battery-powered nodes are called ZigBee *end devices*. End devices typically sleep most of the time, waking only to periodically measure and report data as well as to check for and respond to incoming messages. A third type of node, a ZigBee *router*, is similar to an end device except that it is connected to a consistent power supply (i.e., power outlet) and therefore does not need to sleep. It can store messages for end devices while they are sleeping and can therefore extend the network range. In other words, instead of needing to reach the coordinator's antenna directly, a node may communicate with the nearest router, which then communicates with the next closest router, and so on until the message reaches the coordinator. The resulting self-healing mesh network can have a theoretical range of hundreds of meters and support up to 64,000 devices [3].

4.3.1.2 ZigBee Security

When operating in secure mode, ZigBee uses two Advanced Encryption Standard (AES) encryption keys: a link key and a network key. Like most WSN protocols, ZigBee can only use security keys if they are pre-installed on each node. Otherwise the keys must be distributed by the network in an insecure manner, or not used at all. As long as each node is pre-configured

with the same link key, a ZigBee node can use that key to join the network and securely obtain the current network key.

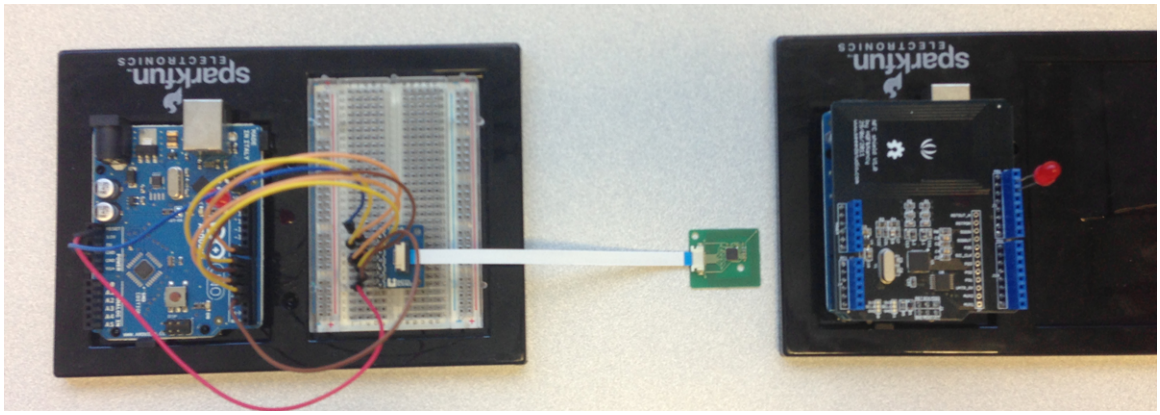
4.3.1.3 Why Implement on ZigBee

A ZigBee network would greatly benefit from the new NFC pairing technique. Pairing would allow nodes to securely obtain the current keys without needing to be configured ahead of time, thus eliminating the need to install firmware with new keys.

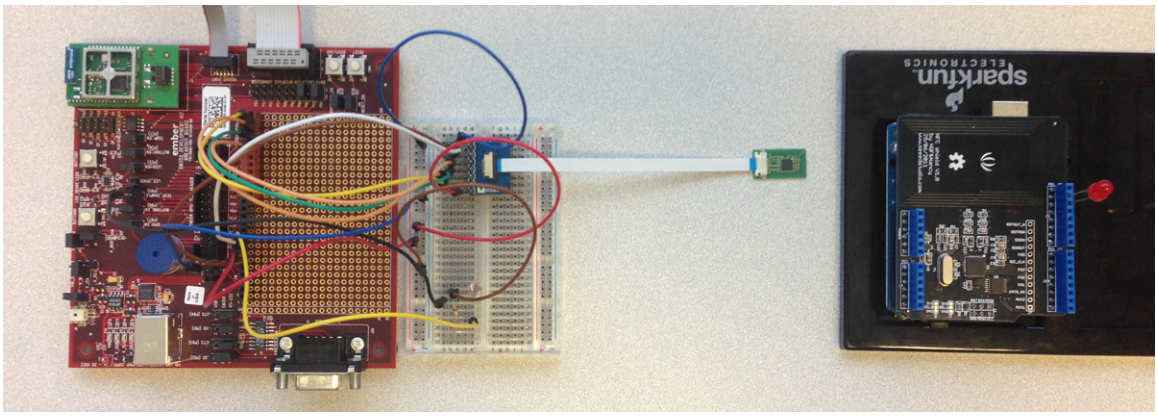
According to ABI Research, "ZigBee comprised roughly 40% of the 2010 IEEE 802.15.4 chipset shipments and will grow to comprise nearly 55% of 2016 IEEE 802.15.4 chipset shipments [2]." Therefore, we chose to integrate the new NFC pairing method into the ZigBee protocol due to its growing popularity, its well-designed WSN architecture, and the potential to improve the standard with secure pairing.

4.3.2 Integrating NFC

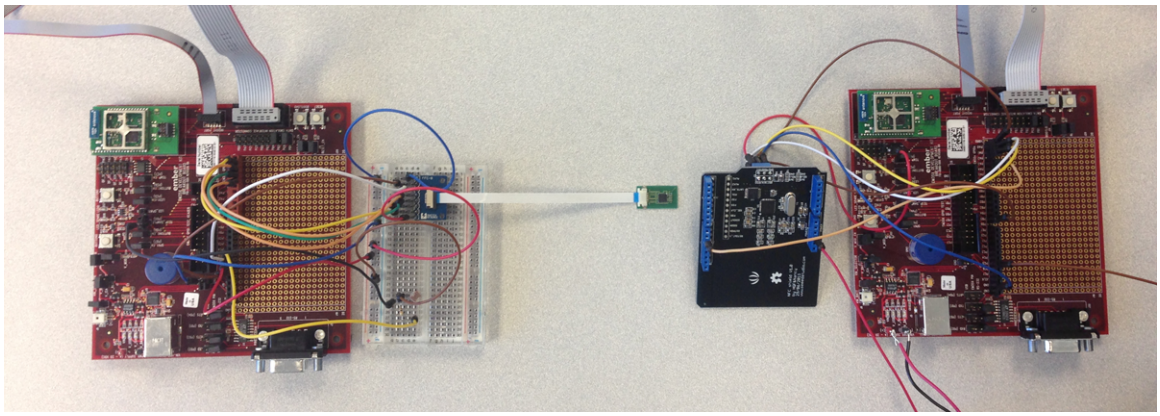
With the NFC shield and the dynamic tag working properly with Arduinos, we made the switch from Arduino to the EM357 as shown in Figure 4.6. The EM35x development kit came with development boards which allowed easy access to all the chip's pins as well as a variety of peripherals. We wired the needed pins from the NFC shield to one EM357, and the dynamic tag to another. We converted all our Arduino code and necessary libraries from Arduino's format to C and converted the hardware-level commands and timers. We tested both the NFC reader/writer and the dynamic tag by interacting via NFC with the working Arduino prototypes, and then with each other. At this point we had NFC hardware for dynamic pairing on separate ZigBee wireless nodes. Figure 4.7 shows the hardware components on both our actuator or sensor node and the coordinator node up to this point.



(a) Left: NFC dynamic tag and Arduino. Right: NFC reader/writer (shield) and Arduino.



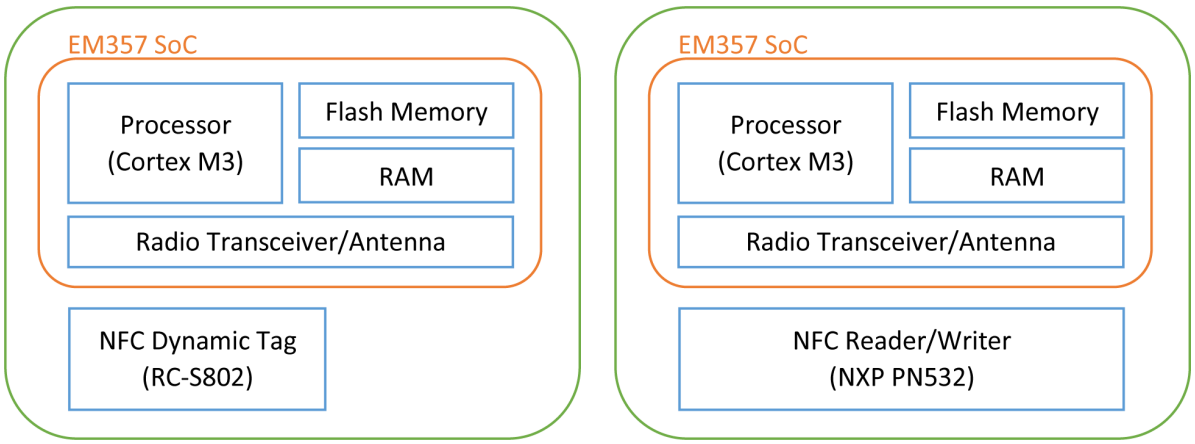
(b) Left: NFC dynamic tag and EM357. Right: NFC reader/writer (shield) and Arduino.



(c) Left: NFC dynamic tag and EM357. Right: NFC reader/writer (shield) and EM357.

Figure 4.6: Transition From Arduino to EM357

The Arduino driving the NFC dynamic tag in (a) was replaced with the EM357 in (b). The Arduino driving the NFC reader/writer in (a) and (b) was replaced with the EM357 in (c). The Arduinos helped us program the NFC hardware in a familiar prototyping environment, but needed to be replaced with a processor that could handle the ZigBee protocol (the EM357).



(a) Sensor or Actuator Node (b) Coordinator Node

Figure 4.7: Nodes with the EM357 SoC

4.3.3 Network Formation

The EM357 comes with a copy of EmberZNet PRO, a proprietary ZigBee-compliant wireless stack with API calls to help abstract many of ZigBee's inner details from wireless application developers. Our task was to modify the stack to incorporate the new dynamic NFC pairing. When using pre-installed link keys, ZigBee adds new nodes to the network following these steps:

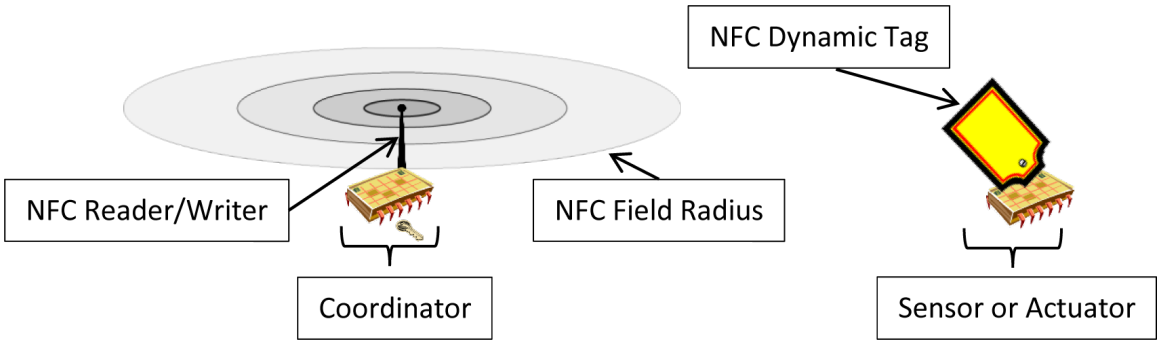
- **IEEE 802.15.4 Beacon:** The coordinator periodically broadcasts an IEEE 802.15.4 and ZigBee combined beacon. Each beacon supplies generic information about the network such as supported protocols, the radio channel, and the network's unique ID. The joining node waits until it receives a beacon. It may send an IEEE 802.15.4 beacon request message to which the coordinator responds with the beacon.
- **IEEE 802.15.4 Association:** The joining node initiates an IEEE 802.15.4 Association. Using the information gathered from the beacon, the joining node sends an Association Request to the coordinator and receives an ACK. The joining node sends a Data Request message and receives an ACK. Lastly, the coordinator sends the Association Response and gets an ACK.

- **ZigBee Transport Key:** The coordinator immediately sends the network key, encrypted using the pre-installed link key, to the joining node and receives an ACK. The joining node is only able to decrypt the network key correctly if it uses the matching pre-installed link key.
- **ZigBee Device Announce:** The joining node immediately sends a Device Announce broadcast, encrypting the message with the received network key. Any nodes on the same network with the correct network key will get the broadcast and may communicate with the node using any of ZigBee's addressing schemes. If the joining node was unable to properly decrypt the network key, the Device Announce message will be garbage and no nodes will respond.

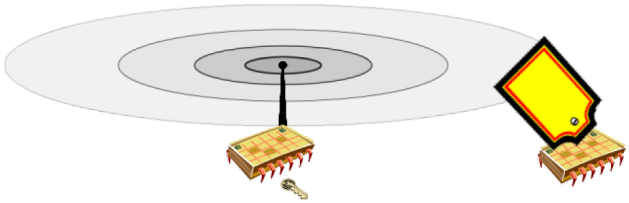
Using EmberZNet PRO's stack callbacks functions, we modified the network joining process to require dynamic NFC pairing. ZigBee's secure mode of operation requires each node to be pre-installed with matching keys. Our implementation does not use any pre-installed keys. Rather, the coordinator transfers the current link key and settings to a joining node using NFC. The joining node then uses the link key, as if it had been pre-installed, to securely join the network. ZigBee already allows nodes with the correct pre-installed keys to join a network, and prevents nodes without the correct pre-installed keys from joining. Since we did not alter ZigBee's behavior regarding the use of pre-installed keys, our modifications left the stack in compliance with ZigBee standards.

The new join procedure adds dynamic NFC pairing prior the Beacon phase as shown in Figure 4.8. The steps to pairing are as follows:

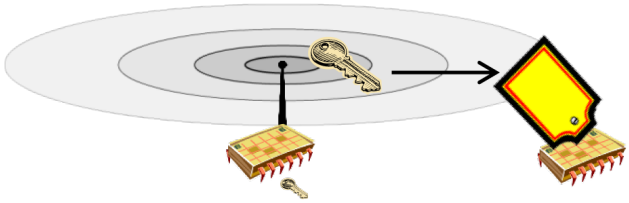
- **Enable NFC:** The joining device enables its NFC dynamic tag by supplying power. The tag enters a low-power mode (Figure 4.8a).
- **Detect NFC RF:** The user moves the joining node close to the coordinator's NFC reader/writer, which is always on. When the dynamic tag detects NFC radio frequency



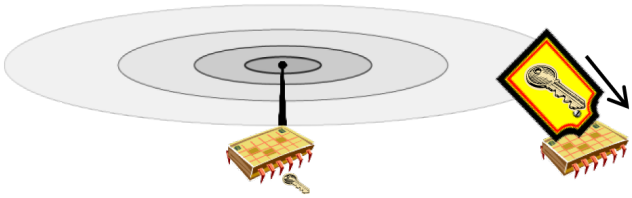
(a) The joining node (sensor or actuator) powers on its NFC tag on start up.



(b) The user brings the coordinator and joining node close together. The dynamic tag senses the NFC RF activity and enters full-power mode. The coordinator's NFC reader/writer polls for NFC tags. Once the dynamic tag is in full-power mode, it responds to the poll with its ID.



(c) The coordinator sends the key and settings to its NFC reader/writer, which writes them to the joining node's dynamic tag.



(d) The joining node reads the key and settings out of its dynamic tag. The joining node uses the key to securely join the network. The joining node powers off the dynamic tag.

Figure 4.8: Dynamic NFC Pairing Implementation

(RF) activity, the joining node supplies the tag with its initial settings. The tag enters full-power mode and is ready to communicate (Figure 4.8b).

- **Poll For Tags:** The coordinator periodically sends an NFC polling message. If a dynamic tag receives the poll, it sends a response message that includes its ID. The coordinator receives the message and *captures* the tag (Figure 4.8b).
- **Write Tag:** The coordinator sends an NFC write command with the link key, network ID, application channel, and any other important settings to the captured tag (Figure 4.8c). The dynamic tag indicates to the joining node's micro-controller that it has received data. The joining node reads the data out of its dynamic tag (Figure 4.8d).

At this point the joining node has everything it needs in order to securely join the wireless network. The regular ZigBee secure join procedure takes place. Once joined to the network, the node turns off its NFC dynamic tag to conserve power.

4.3.4 Sample Application

As a demonstration, we developed an application which uses dynamic NFC pairing to add nodes to a network. The network coordinator node has an integrated NFC reader/writer and forms the ZigBee network. All other nodes have NFC dynamic tags and join the network automatically when held close to the coordinator. We created 3 sensor nodes: two light sensors and a temperature sensor. These sensors report light level and temperature measurements to the coordinator every few seconds. We also created 2 actuator nodes: a music player and a power switch. Our application allows the coordinator to send commands to the actuators to play the music and to turn on or off the switch. For example, if there is a string of Christmas tree lights plugged into the power switch, then the coordinator can tell the switch actuator to turn the lights on and off.

4.4 Cloud

A physical demonstration in which wireless nodes pair securely is not visually astounding because you cannot see a wireless pairing; at most a light blinks. A demonstration where sensors collect data and report them wirelessly to a coordinator is also invisible. To make our work tangible and recognizable, we went one step further and connected the network to the Internet. This allows us to stream the live sensor data to a web browser and control the sensor network with a mouse.

This section describes the elements in our system needed to allow the user to control a WSN using a web browser. Specifically, we add a WiFi module to the WSN coordinator and configure a remote web server.

4.4.1 WiFi Module

The logical node to turn into an Internet gateway is the network coordinator, since it already gathers all the sensor data. So, we added a WiFi module, the RN-131C WiFly GSX, to the EM357 that was already connected to the NFC reader/writer. The WiFi module allows us to connect the WSN to the Internet via WiFi and make HTTP requests to our cloud server. The coordinator can then post sensor readings and poll the server for network commands. The final coordinator hardware, after integrating the WiFi module, is shown in Figure 4.9.

Ideally, there would be a persistent open connection such as a websocket between the coordinator and the cloud server so that data could be reported immediately in an event-driven architecture rather than by polling. An alternative would be to assign the coordinator a public IP address so the server could send commands at any time. Unfortunately, this would require the user to set up port forwarding or to obtain a static IP address for the WSN. Either way, this would add complications for the end user. Ultimately, we implemented a polling system due to the lack of time and because event-driven architectures is not the purpose of our demonstration.

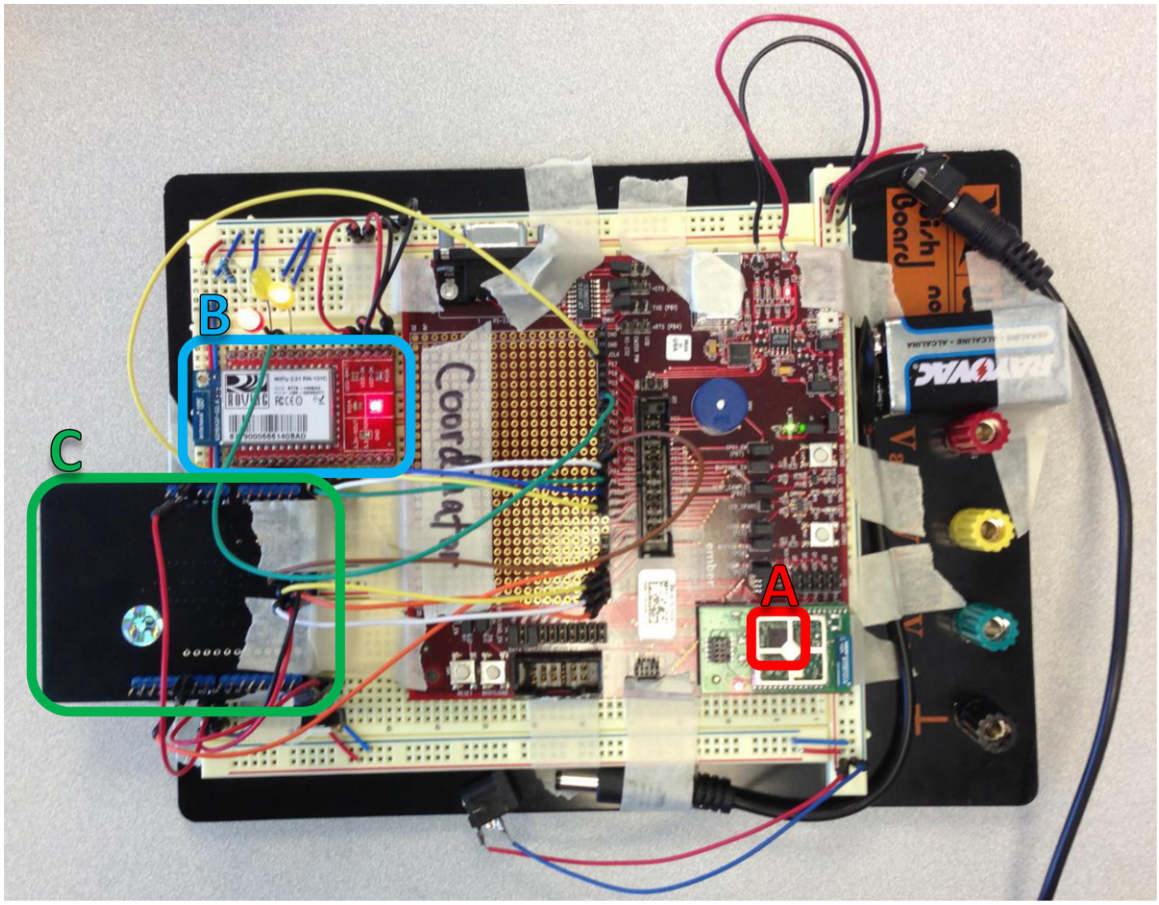


Figure 4.9: Final Coordinator Hardware

The EM357 (A), WiFi module (B), and NFC reader/writer (C) connected together to create a node that is both a WSN coordinator and an Internet gateway.

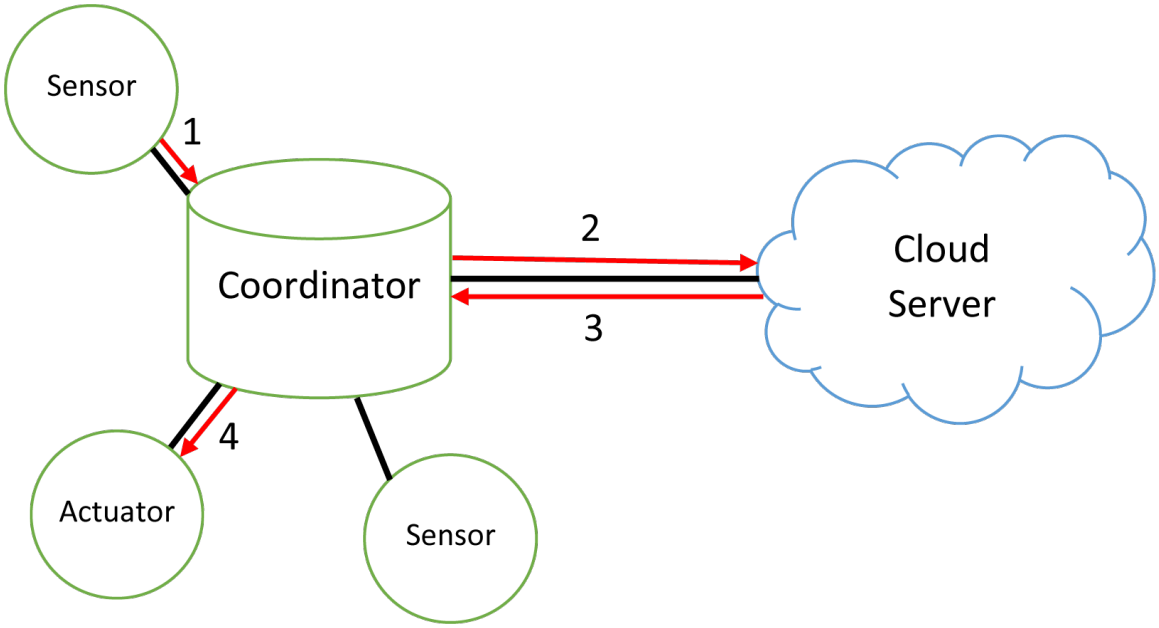


Figure 4.10: Example Data Flow in WSN

(1) A light sensor reports the current light reading to the coordinator. (2) The coordinator posts the light reading to the cloud server. (3) The cloud server determines that the reading is too dark and sends the coordinator a command to turn on the lamp. (4) The coordinator tells the lamp actuator to turn on the lamp.

4.4.2 Server Implementation

In our implementation, we set up a remote web server with a static IP address and a public domain name. The server and WSN coordinator work together as shown in Figure 4.10. Sensors constantly report data about their environment to the coordinator. The coordinator consolidates all the sensor data and periodically sends it to our cloud server. When the server determines that an action should occur on the the WSN, the server sends a command to the coordinator. The coordinator passes the command to the relevant actuator. The actuator receives and performs the command. We now explain the technical details of this process.

4.4.2.1 Server Back-End

We implemented an endpoint for the coordinator to report data in an HTTP post. The post contains a JSON-encoded message of sensor readings and the status of the actuators. The server updates a MySQL database to match the received data, adding new records for

new devices and removing old records from removed devices. In the HTTP response, the server returns a list of queued commands for the sensor network. The coordinator parses the commands and sends each command to its affiliated actuator. This means that, in this demonstration, commands from the server will not arrive at the WSN until the coordinator posts to the cloud server. For this reason, the coordinator posts to the server frequently (every second), even when no sensor readings have changed.

It is important to note that in this demonstration, all sensor command logic is maintained in the cloud server. A WSN without an Internet gateway must trust the coordinator to decide when events should occur; for example, turn on the lamp when the light level is too dark. However, in our system, the coordinator does not contain this logic; the cloud server does. In other words, the WSN application does nothing more than report data to the server and execute commands received from the server. The server and the user together determine the logic behind commands. For example, if a user wants to turn on a lamp any time the light level gets too dark, the server would get continual light readings from the coordinator, decide when it gets too dark, and then tell the coordinator to turn on the lamp.

This new WSN architecture is extremely flexible and dynamic. The user may add nodes to the WSN at any time. When a node is added, the coordinator reports the change to the cloud server. The user can then program or configure the server to act however he desires. The WSN application firmware on the physical nodes never needs to change. Web programmers can now build and share WSN applications through the cloud without ever touching hardware or firmware.

4.4.2.2 User Interface

The server displays a simple web page to the user that lists all their WSN data in (almost) real-time as shown in Figure 4.11. The page uses AJAX to continually poll the MySQL database and refresh sensor and actuator readings on the web page that need to be updated,

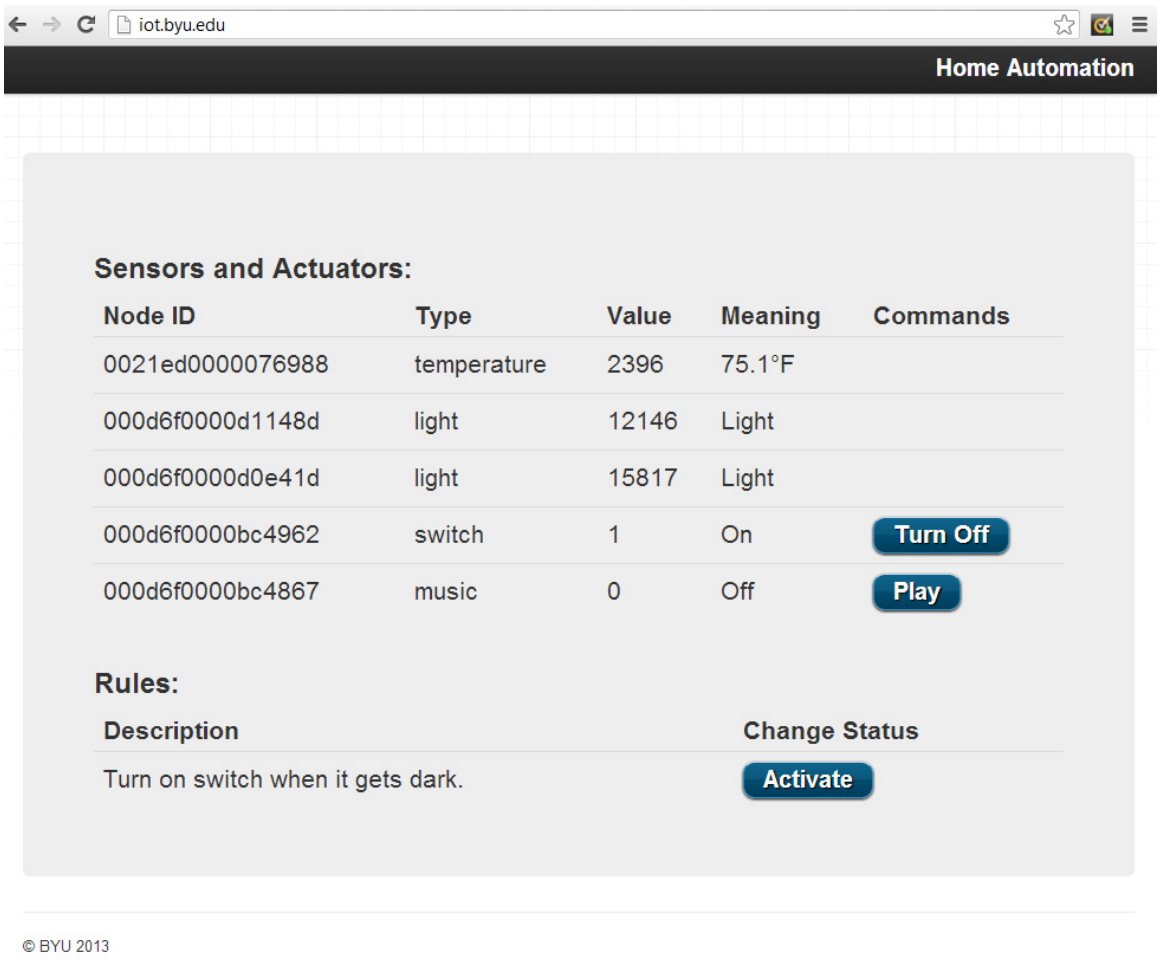


Figure 4.11: Web Browser User Interface

The web browser shows the user the status of the sensors and actuators. Buttons allow the user to cause events to occur on the WSN.

thus giving the illusion of data streaming. We also added control buttons to the web page which allow the user to cause events to occur on the sensor network, such as turning on switches, or playing music. This allows the end user to see the potential of dynamic NFC pairing: all the user had to do was to hold each wireless node close to the coordinator for a moment, and now he is almost instantly able to view and control the whole network from practically anywhere in the world via the Internet.

Chapter 5

Results

We now evaluate the performance of the sample ZigBee application which uses dynamic NFC pairing. We compare it with a version of the same application that uses pre-installed keys instead of pairing. Note that these results are specific to the ZigBee stack implementation EmberZNet PRO version 5.0.

5.1 Security

Since both applications use security keys in the same way, they are both secure during operation. However, dynamic NFC pairing is arguably more secure than using pre-installed keys because keys never reside in non-volatile flash memory. Therefore, keys are unknown to the manufacturer and cannot be discovered by probing a node before network formation (see Section 3.3.2 for more details). Pairing avoids the potential security problems of using key pre-installation because the key is transferred only at the moment a node joins the network.

It is important to remember that although a network may operate safely from wireless attacks, there are physical attacks in which an attacker could still obtain a network key. For example, an attacker could perform a cold boot attack in which he physically freezes a node with liquid nitrogen and then probes the RAM with an oscilloscope to discover the keys [12]. However, the use of ZigBee security keys is generally accepted as being "safe."

5.2 Usability

The application with pre-installed keys is more convenient and usable to the user because he doesn't have to do anything to add nodes to his network; nodes automatically join. We saw earlier that of all the pairing methods, NFC is one of the most usable and least expensive options while still remaining secure. The user involvement is kept at a minimum, only requiring the user to hold a node close to the coordinator.

5.2.1 Time of Pairing

Adding NFC to the network coordinator does not affect network formation time because the coordinator scans the radio channels, selects one, and forms a network automatically. In our application, this takes about 5.9 seconds with and without dynamic NFC pairing. The EmberZNet PRO stack allows developers to adjust the scan time to meet an application's needs. Longer scan times allow the coordinator to more accurately choose the channel with the least noise on which to form its network.

Once the coordinator is running and has formed the network, it takes little time for a wireless node to join. Although the time needed to join the network may vary due to network traffic, interference, and other factors, we measured the time to discover and join our network to be roughly 950ms when using pre-installed keys. Our measurements show that when dynamic NFC pairing is used, it takes about 2ms to transfer a key using NFC. However, before the key transfer may begin, the NFC reader/writer must discover the NFC dynamic tag by polling. In our application we poll for tags every 500ms, meaning that the time to discover a tag once it close to an NFC reader/writer ranges from 0 to 500ms. Therefore, the worst case join time for our application with dynamic NFC pairing is about 1452ms. We assume that the additional 502ms duration is small enough for users to tolerate. Application developers may modify the poll time as needed.

5.3 Cost

We evaluate the cost in terms of energy consumption, memory usage, and hardware expense.

5.3.1 Energy Consumption

In this section we evaluate the energy consumption of ZigBee sleepy end devices. The EM357 draws a current of 31mA during transmission, 26mA while receiving a message, and 800nA when sleeping with a timer. We assume that a sleepy sensor wakes up every t seconds, reports its sensor data, and goes back to sleep. Measurements on the sample application show that sleepy nodes remain awake for an average of 12ms: about 10ms to report its sensor data, about 1.5ms to receive the ACK, and about 0.5ms to decide to return to sleep. We use these numbers for our energy calculations.

As an example, say we want to power a sleepy sensor that reports its data once every second using AA batteries. The EM357 requires a 2.1-3.6V input to operate. Typical AA batteries produce 1.5V output and have about a 2500mAh capacity. Therefore, if we were to attempt to power a sleepy sensor node with AA batteries, we would need two batteries to get a 3V output. The theoretical battery life of the sensor would be:

$$\frac{2500\,mAh}{\frac{31.17\,mA*12\,ms}{12\,ms+1\,sec*1000\,ms/sec}+\frac{0.0008\,mA*1\,sec*1000\,ms/sec}{12\,ms+1\,sec*1000\,ms/sec}} = 6750\,hours$$

We see that the sensor would run for 6750 hours, or 281 days. Appendix A shows results from using the same equation with different inputs. Appendix A.4 shows battery life given a variety of 3V batteries and differing node sleep cycles. One important result we see is that a sleepy sensor running on a tiny 3V, 30mAh watch battery can last for years if it reports data only a few times an hour.

5.3.1.1 Effect of Pairing

We now evaluate the effect of dynamic NFC pairing on a sleepy sensor. We saw in Section 5.2.1 that the join process using dynamic NFC pairing takes about 1452ms in the worst case: 500ms for the NFC reader/writer to find the dynamic tag, 2ms for the key transfer, and 950ms to complete the join. The NFC dynamic tag only needs to be on long enough to complete the key transfer. Therefore, if it takes the user 10 seconds to move two nodes close together, then the NFC dynamic tag may turn off after 10.502 seconds even though the complete join process will take an additional 950ms.

If we assume that the user takes a generous 30 seconds to hold the node close to the coordinator after turning if on, the node will need to power the NFC dynamic tag for 30.502 seconds. In our implementation, we powered the dynamic tag by connecting one of the EM357's output pins to the tag's input voltage pin. The application then turns off the tag entirely once the key exchange is complete. The RC-S802 FeliCa plug requires 1 mA in operation mode and 1 uA in standby mode. The dynamic tag only switches into operation mode when it detects NFC RF activity, which is just before pairing. So, the tag will need 1 uA for 30.5 seconds, and 1 mA for 2 ms. This depletes only $9.03 * 10^{-6}$ mAh. In other words, it requires so little energy that even tiny batteries won't be noticeably affected.

5.3.2 Memory Usage

Table 5.1 shows the RAM required by the sample application with and without dynamic NFC pairing. Table 5.2 shows the flash storage necessary for both applications. We see that the memory footprint is small, requiring only about 112 B of additional RAM and around 2 kB of extra flash storage for the image.

5.3.3 Hardware Expense

Hardware cost is difficult to approximate because of fluctuating prices as well as the effect of mass-producing electronics over time. However, to give a rough estimate of cost, we look at

	No Pairing	NFC Pairing	Increase
Coordinator	10808 B	10912 B	104 B
Light Sensor	8016 B	8128 B	112 B
Thermometer	7984 B	8096 B	112 B
Music Player	8000 B	8112 B	112 B
Power Switch	7984 B	8096 B	112 B

Table 5.1: RAM Usage

	No Pairing	NFC Pairing	Increase
Coordinator	134180 B	135932 B	1752 B
Light Sensor	115948 B	117600 B	1652 B
Thermometer	120760 B	122424 B	1664 B
Music Player	120444 B	122104 B	1660 B
Power Switch	120668 B	122332 B	1664 B

Table 5.2: Flash Usage

today's prices of purchasing hardware in small quantities. A complete NFC reader/writer prototyping shield costs as little as $25, and only one is needed per network. A single NFC dynamic tag from Sony is currently about $6.50, though other companies are currently building their versions of the same hardware which will be released this year and may drive competition. For our demo network of 1 coordinator and 5 sensors, the hardware cost of adding NFC would therefore be about $57.50, or approximately $10 per node. Of course, as you purchase more sensors, the average cost of adding dynamic NFC pairing per node approaches the price of the dynamic tag.

Currently, ZigBee nodes such as temperature sensors and on/off switches sell for as little as $25 each. If a dynamic tag were added to one of these sensors, the price might raise to $31.50 raising the price by 26%. The cost of regular NFC tags is currently as low as $0.20 per tag [22]. We assume that the dynamic tag will continue to quickly decline in price until it is almost as inexpensive as regular tags. If dynamic tags dropped to 20 cents, the price of a $25 ZigBee sensor would increase by only 0.8% making the pairing method affordable.

Chapter 6

Conclusion

Until now, WSNs have only offered secure communication when nodes are pre-installed with network keys. Key pre-installation, however, requires technical expertise that almost exclusively belongs to manufacturers and trained technicians. Therefore, average users have been unable to set up and control their own WSNs.

This thesis presented dynamic NFC pairing, a new wireless pairing method well-suited for WSNs. By giving the network coordinator node an NFC reader/writer, other nodes can join the network using NFC dynamic tags. Users can easily add nodes to a WSN by simply holding them close to the network coordinator.

Dynamic NFC pairing is secure, even more so than using key pre-installation. The cost of the required NFC hardware is quickly dropping and becoming inexpensive. The pairing method is usable because the user just brings nodes close together and the nodes quickly pair on their own.

To demonstrate the power of dynamic NFC pairing, a sample application was built. The application allows a non-technical user to form their own WSN network by simply holding nodes close to a coordinator node. Sensor readings instantly stream to a web page where the user can view data as well as click buttons that cause events to occur on the network.

6.1 Applications For Dynamic NFC Pairing

We suggest that dynamic NFC pairing is ideal for any scenario in which security is of critical importance, or in which the system must be easy for the user to set up. One example would

be setting up a custom security or automation network in the home. The user wouldn't need to know how to install keys or have to do any complicated pairing button tricks to make things work. Business owners could benefit in the same way. Another application could be a sensor network in a military environment. Sensors could be mailed or shipped without the concern of someone capturing keys in transit. Once the sensors arrive in the field, they could be easily and securely set up without a wireless specialist present.

6.2 Limitations

Dynamic NFC pairing is not suitable for every situation. Although secure and usable, NFC pairing does add some additional cost to wireless devices. In situations in which the user has enough expertise to program his own keys and has access to the WSN nodes' firmware, it may be better not to use NFC pairing. Also, it is possible that NFC may not always work if the 13.56 MHz band is heavily used or jammed, in which case the user would want to use key pre-installation instead.

Even though dynamic NFC pairing establishes cryptographic keys between two nodes, this does not guarantee that an application will be secure. A poorly-written application could expose the keys or even disregard them entirely. It is the responsibility of the application developer to ensure that appropriate safety precautions are taken.

6.3 Future Work

The sample application developed and described in this work is just one example of the benefits of dynamic NFC pairing. At this point, the prototype hardware could be reduced in size and fabricated. The software could be used as the basis for commercial WSN applications. The cloud server acts only as an example but could be used as a model for building a full-scale event-driven engine for WSNs. In short, all the ground work for a commercial application using dynamic NFC pairing is done. Future work could involve manufacturing and distributing the system.

References

[1] ABIResearch. 850 million IEEE 802.15.4 chipsets to ship in 2016, despite strong competition from bluetooth, May 2012. URL `http://www.abiresearch.com/press/850-million-ieee-802154-chipsets-to-ship-in-2016-d`.

[2] ZigBee Alliance. Market leadership, 2013. URL `http://www.zigbee.org/About/AboutTechnology/MarketLeadership.aspx`.

[3] ZigBee Alliance. Zigbee standards overview, January 2013. URL `http://www.zigbee.org/Standards/Overview.aspx`.

[4] J. Suomalainen; J. Valkonen; N. Asokan. Security associations in personal networks: A comparative analysis. *Security and Privacy in Ad-hoc and Sensor Networks: Lecture Notes in Computer Science*, pages 43–57, 2007.

[5] N. Saxena; J.E. Ekberg; K. Kostiainen; N. Asokan. Secure device pairing based on a visual channel. *Security and Privacy, 2006 IEEE Symposium on*, pages 6–313, 2006.

[6] S.A. Camtepe and B. Yener. Combinatorial design of key distribution mechanisms for wireless sensor networks. *Networking, IEEE/ACM Transactions on*, 15(2):346–358, April 2007. ISSN 1063-6692. doi: 10.1109/TNET.2007.892879.

[7] K. Rasmussen; C. Castelluccia; T. Heydt-Benjamin; S. Capkun. Proximity-based access control for implantable medical devices. *CCS '09 Proceedings of the 16th ACM Conference on Computer and Communications Security*, 2009.

[8] I.F. Akyildiz; W. Su; Y. Sankarasubramaniam; E. Cayirci. Wireless sensor networks: A survey. *Broadband and Wireless Networking Laboratory, School of Electrical and Computer Engineering, Georgia Institute of Technology*, 2001.

[9] Uday B Desai, BN Jain, and SN Merchant. Wireless sensor networks: Technology roadmap. White Paper, April 2010. Ministry of Information and Communication Technology, India.

[10] M. Eltoweissy, M. Younis, and K. Ghumman. Lightweight key management for wireless sensor networks. In *Performance, Computing, and Communications, 2004 IEEE International Conference on*, pages 813 – 818, 2004. doi: 10.1109/PCCC.2004.1395190.

[11] K. Nyberg; C. Gehrmann. Enhancements to bluetooth baseband security. *Proceedings of Nordsec*, 2001:191–230, 2001.

[12] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, May 2009. ISSN 0001-0782. doi: 10.1145/1506409.1506429. URL `http://doi.acm.org/10.1145/1506409.1506429`.

[13] Shih-I Huang, Shiuhpyng Shieh, and S.Y. Wu. Adaptive random key distribution schemes for wireless sensor networks. In D.T. Lee, S.P. Shieh, and J.D. Tygar, editors, *Computer Security in the 21st Century*, pages 91–105. Springer US, 2005. ISBN 978-0-387-24005-3. doi: 10.1007/0-387-24006-3_7. URL `http://dx.doi.org/10.1007/0-387-24006-3_7`.

[14] M. Cagalj; S. Capkun; J.P. Hubaux. Key agreement in peer-to-peer wireless networks. *Proceedings of the IEEE*, 94(2):467–478, 2006.

[15] Sajid Hussain, Firdous Kausar, and Ashraf Masood. An efficient key distribution scheme for heterogeneous sensor networks. In *Proceedings of the 2007 international conference on Wireless communications and mobile computing*, IWCMC '07, pages 388–392, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-695-0. doi: 10.1145/1280940.1281023. URL `http://doi.acm.org/10.1145/1280940.1281023`.

[16] Cynthia Kuo, Jesse Walker, and Adrian Perrig. Low-cost manufacturing, usability, and security: An analysis of bluetooth simple pairing and wi-fi protected setup. In Sven Dietrich and Rachna Dhamija, editors, *Financial Cryptography and Data Security*, volume 4886 of *Lecture Notes in Computer Science*, pages 325–340. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-77365-8. doi: 10.1007/978-3-540-77366-5_30. URL `http://dx.doi.org/10.1007/978-3-540-77366-5_30`.

[17] Ming Li, Shucheng Yu, Wenjing Lou, and Kui Ren. Group device pairing based secure sensor association and key management for body area networks. In *INFOCOM, 2010 Proceedings IEEE*, pages 1 –9, march 2010. doi: 10.1109/INFCOM.2010.5462095.

[18] C. Gehrmann; K. Nyberg. Security in personal area networks. *In C. J. Mitchell, editor, Security for Mobility*, pages 191–230, 2004.

[19] C. Mitchell; C. Gehrmann; K. Nyberg. Manual authentication for wireless devices. *Cryptobytes*, page 7, 2004.

[20] L.B. Oliveira, H.C. Wong, M. Bern, R. Dahab, and A.A.F. Loureiro. Secleach - a random key distribution solution for securing clustered sensor networks. In *Network Computing and Applications, 2006. NCA 2006. Fifth IEEE International Symposium on*, pages 145 –154, july 2006. doi: 10.1109/NCA.2006.48.

[21] Leonardo B. Oliveira, Diego F. Aranha, Conrado P.L. Gouva, Michael Scott, Danilo F. Cmara, Julio Lpez, and Ricardo Dahab. Tinypbc: Pairings for authenticated identity-based non-interactive key distribution in sensor networks. *Computer Communications*, 34(3):485 – 493, 2011. ISSN 0140-3664. doi: 10.1016/j.comcom.2010.05.013. URL `http://www.sciencedirect.com/science/article/pii/S0140366410002483`. Special Issue of Computer Communications on Information and Future Communication Security.

[22] RapidNFC. Can nfc create the internet of things?, May 2013. URL `http://rapidnfc.com/blog/75/can_nfc_create_internet_of_things`.

[23] J.M. McCune; A. Perrig; M.K. Reiter. Seeing-is-believing: Using camera phones for human-verifiable authentication. *Security and Privacy, 2005 IEEE Symposium on*, pages 110–124, 2005.

[24] R. Kainda; A. Flechais; A.W. Roscoe. Usability and security of out-of-band channels in secure device pairing protocols. *Proceedings of the 5th Symposium on Usable Privacy and Security*, 2009.

[25] Nickolai Zeldovich Shyamnath Gollakota, Nabeel Ahmed and Dina Katabi. Secure in-band wireless pairing. *USENIX Security Sym.*, 2011.

[26] A. Kumar; N. Saxena; G. Tsudik; E. Uzun. A comparative study of secure device pairing methods. *Pervasive Computing and Communications. IEEE International Conference on*, pages 1–10, 2099.

[27] C. Soriente; G. Tsudik; E. Uzun. Beda: Button-enabled device association. 2007.

[28] Ersin Uzun, Kristiina Karvonen, and N. Asokan. Usability analysis of secure pairing methods. In Sven Dietrich and Rachna Dhamija, editors, *Financial Cryptography and Data Security*, volume 4886 of *Lecture Notes in Computer Science*, pages 307–324. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-77365-8. doi: 10.1007/978-3-540-77366-5_29. URL `http://dx.doi.org/10.1007/978-3-540-77366-5_29`.

[29] S. Vaudenay. Secure communications over insecure channels based on short authenticated strings. *Advances in Cryptology-CRYPTO*, pages 309–326, 2005.

[30] A. Kobsa; R. Sonawalla; G. Tsudik; E. Uzun; Y. Wang. Serial hook-ups: A comparative usability study of secure device pairing methods. *Proceedings of the 5th Symposium on Usable Privacy and Security*, 2009.

[31] D.B. Smetters; D. Balfanz; D.K. Smetters; P. Stewart; H.C. Wong. Talking to strangers: Authentication in ad-hoc wireless networks. 2002.

Appendix A

Energy Calculations

The EM357 draws a current of 31 mA during transmission, 26 mA while receiving a message, and 800nA when sleeping with a timer. We assume that a sleepy sensor wakes up every t seconds, reports its sensor data, and goes back to sleep. From the measurements taken from our sample application, we determine that sleepy nodes remain awake for an average of 12ms: about 10ms to report its sensor data, about 1.5 ms to receive the ACK, and about 0.5ms to decide to return to sleep. These are the numbers used in our calculations below.

The EM357 was designed to allow sleepy nodes to wake up and go to sleep extremely quickly (110 us to wake; 5 us to sleep). Therefore, we ignore wake up and shutdown times in our calculations. To simply calculations, we will assume 10 ms of transmission and 2 ms of reception for each wake cycle. Therefore, the average current during the 12 ms wake duration is 31*10/12 + 26*2/12 = 30.17 mA. The current needed during sleep is 800 nA or 0.0008 mA.

It is important to note that the EM357 requires a 2.1-3.6V input for operation. Therefore, when considering the life of a battery, the battery must first provide the correct voltage output.

A.1 Battery Life Equation

Below is the equation for approximating the theoretical maximum battery life given a battery capacity and duration of node sleep cycle.

$$c = battery\,capacity\,(mAh)$$

$$a = 31.17mA = average\,mA\,when\,node\,is\,awake$$

$$d = 12ms = \ time\,awake\,in\,one\,wake\,cycle\,(ms)$$

$$s = 0.0008mA = average\, mA\, when\, node\, is\, sleeping$$

$$t = \ time\, sleeping\, in\, one\, wake\, cycle\, (s)$$

$$\frac{c}{\frac{a*d}{d+t*1000} + \frac{s*t*1000}{d+t*1000}} = hours\, of\, battery\, life$$

Constraining the input to match our application, we get:

$$\frac{c}{\frac{31.17*12}{12+t*1000} + \frac{0.0008*t*1000}{12+t*1000}} = hours\, of\, battery\, life$$

A.2 c=2500, t=1

As an example, we attempt to power a sensor that reports every 1 second with AA batteries. One AA battery supplies 1.5V and 2500mAh, so we will need two AA batteries to produce a 3V input. The battery life equation gives us:

$$\frac{2500}{\frac{31.17*12}{12+1*1000} + \frac{0.0008*1*1000}{12+1*1000}} = 6750\, hours\, of\, battery\, life$$

We see that the theoretical maximum for the sensor reporting every second on two AA batteries is 6750 hours, or 281 days.

A.3 c=30, t=60

Typically, WSN nodes ship with small batteries, much smaller than AAs. Let's say we power the sleepy node with tiny 3V watch battery rated at only 30mAh, and that the node only reports its reading once a minute.

$$\frac{30}{\frac{31.17*12}{12+60*1000} + \frac{0.0008*60*1000}{12+60*1000}} = 4272\, hours\, of\, battery\, life$$

In this case, the sensor lasts for 4272 hours, or 178 days.

A.4 Battery Life Table

Table A.1 shows a table that demonstrates the how battery life (days) is affected by battery capacity (mAh) and the sensor's sleep cycle (seconds). The various battery capacity numbers are all actual 3V battery capacities on the market, except for 2500mAh which comes from using 2 AA batteries. Note that these calculations are theoretical maximums and do not take into consideration time decay of battery chemicals.

Sleep Cycle (s) \ Cap. (mAh)	30	40	57	75	85	160	220	280	600	1000	2500
1	3	4	6	8	10	18	25	31	67	112	281
5	17	22	31	41	47	88	122	155	331	552	1381
10	33	44	62	82	93	175	240	306	655	1092	2730
30	94	126	179	236	267	503	691	880	1885	3142	7854
60	178	237	338	444	504	948	1303	1659	3555	5925	14812
300	611	814	1160	1527	1730	3257	4479	5700	12215	20358	50894
600	878	1171	1669	2195	2488	4684	6440	8197	17564	29273	73183
1800	1240	1654	2357	3101	3514	6615	9096	11576	24807	41344	103361
3600	1383	1844	2628	3457	3918	7375	10141	12907	27658	46097	115242

Table A.1: Theoretical Battery Life in Days

Table A.2 shows the same results in years instead of days.

Sleep Cycle (s) \ Cap. (mAh)	30	40	57	75	85	160	220	280	600	1000	2500
1	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.2	0.3	0.8
5	0.0	0.1	0.1	0.1	0.1	0.2	0.3	0.4	0.9	1.5	3.8
10	0.1	0.1	0.2	0.2	0.3	0.5	0.7	0.8	1.8	3.0	7.5
30	0.3	0.3	0.5	0.6	0.7	1.4	1.9	2.4	5.2	8.6	21.5
60	0.5	0.6	0.9	1.2	1.4	2.6	3.6	4.5	9.7	16.2	40.6
300	1.7	2.2	3.2	4.2	4.7	8.9	12.3	15.6	33.5	55.8	139.4
600	2.4	3.2	4.6	6.0	6.8	12.8	17.6	22.5	48.1	80.2	200.5
1800	3.4	4.5	6.5	8.5	9.6	18.1	24.9	31.7	68.0	113.3	283.2
3600	3.8	5.1	7.2	9.5	10.7	20.2	27.8	35.4	75.8	126.3	315.7

Table A.2: Theoretical Battery Life in Years