Secure Over-The-Air Firmware Updates for Sensor Networks

Kevin Kerliu
Dept. of Electrical Engineering
The Cooper Union
New York, NY 10003, USA
kerliu@cooper.edu

Alexandra Ross
Dept. of Computer Science
Johns Hopkins University
Baltimore, MD 21218, USA
alross466@gmail.com

Gong Tao, Zelin Yun,
Zhijie Shi, Song Han, Shengli Zhou
University of Connecticut
Storrs, CT 06269, USA
{tao.gong, zelin.yun, zshi, song.han, shengli}@uconn.edu

***Abstract*—Large-scale wireless sensor networks have been used in the Internet of Things (IoT) and other cyber-physical systems (CPS) to collect large amounts of data. One important practical issue in large-scale sensor networks is firmware updates as updating many deployed sensors one by one is tedious. In this work, we study secure over-the-air (OTA) firmware updates for large-scale wireless sensor networks. We have designed a prototype system where a node can update the firmware in all nearby sensor nodes simultaneously. A custom bootloader is built for sensor nodes to either enter the update mode or boot into existing firmware. A one-way wireless protocol is designed to broadcast new firmware from one node to many other nodes. A shared key among the sensor nodes is used to encrypt/decrypt data packets and for authentication. The prototype system has been tested in ideal and non-ideal environments, where a node can receive the entire firmware in 2 and 5 rounds of transmissions, respectively.**

I. Introduction

Large-scale wireless sensor networks have received more and more attention because of their important roles in the Internet of Things (IoT) and other cyber-physical systems. They cover large physical areas and collect huge amounts of data, which are essential in many control systems, and to big data analysis and machine learning algorithms. Testbed systems are also created to facilitate research in the area. For example, Figure 1 shows the topology of a testbed deployed in the ITE building at the University of Connecticut (UConn). It consists of many sensor nodes that communicate with each other in the IEEE 802.15.4e TSCH mode. The testbed supports multi-hop data forwarding.

One practical issue in large-scale sensor networks is firmware updates. Currently, updating firmware is time-consuming and tedious. Each device must be taken down from its deployment location, connected to a computer to update firmware, and then reinstalled. Over-the-air (OTA) updates are inherently more efficient and increase the scalability of the system significantly.

OTA updates have been around for some time. For example, many cell phones support OTA updates of their operating system and applications; reference [3] discussed OTA updates for automotive industry. Recently, a new platform was designed to better support OTA updates over long ranges

Kerliu and Ross were supported by the NSF grant (REU site) 1659764.

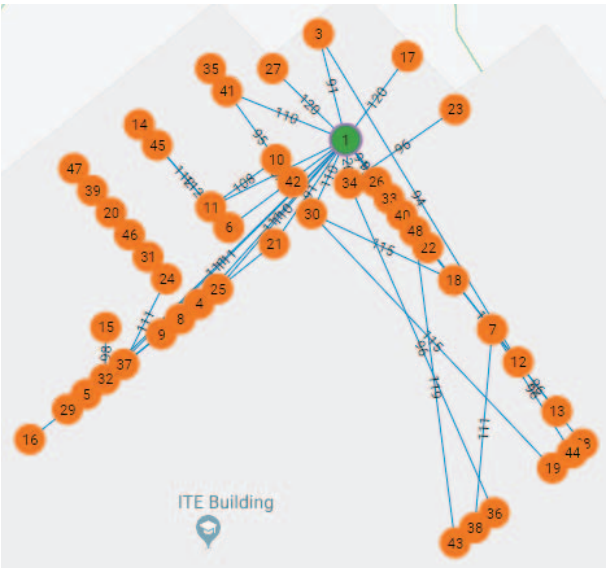

Fig. 1. Topology of a sensor network testbed at UConn

[4]. However, updating firmware in sensor networks faces unique challenges. Sensor nodes have limited resources, e.g., computational power and storage. The update process must be more automated because the sensor nodes have drastically less user interaction abilities than larger devices like cell phones. Lastly, the update process needs to support updating many devices simultaneously as it is not efficient to update many individual nodes one by one.

Ideally, the firmware update process can be initiated from a node and the new firmware is propagated through the entire network automatically. In this paper, however, we study a basic setup. Instead of updating the entire sensor network, a device with the new firmware, referred to as the transmitter, only updates multiple sensor nodes that can receive the firmware from the transmitter directly. These nodes, which can get the firmware from the transmitter, are referred to as the receiver nodes. Figure 2 illustrates a system where a transmitter node can update three receiver nodes through wireless channel simultaneously.

978-1-7281-4121-3/19/$31.00 ©2019 IEEE
DOI 10.1109/MASSW.2019.00026

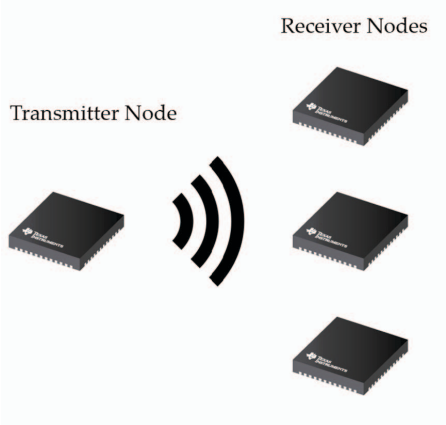

Fig. 2. Transmitter and receiver nodes

The remainder of the paper is structured as follows. We describe our approach and techniques in Section II. In Section III, we discuss the results collected from this project. Finally, we briefly summarize key ideas and discuss future work.

II. Design

As mentioned in Section I, our work focuses on the OTA firmware update issue in a system illustrated in Figure 2, where one transmitter node updates all nearby receiver nodes simultaneously through a wireless channel. The transmitter node has the new firmware at the beginning of the update process. When all the receiver nodes are placed in the update mode, the transmitter node broadcasts the new firmware repeatedly until all receiver nodes have received the new firmware correctly. The main task in our design is 1) to build a custom bootloader for receiver nodes that supports both OTA updates and the normal booting process, and 2) to design a protocol that allows receiver nodes to receive new firmware reliably from the transmitter. Many of the design decisions are dependent on the resources available in sensor nodes. In this section, we will first describe the sensor node we use and then our design.

A. Sensor Nodes

The sensor nodes we are working with are based on Texas Instruments (TI)'s CC2650 lightweight microcontroller. Figure 3 shows the main components that are relevant to this project. The CC2650 microcontroller has an ARM Cortex-M3 processor equipped with 128 KB of flash memory, 8 KB of cache, and 20 KB of SRAM. The flash memory is divided into 4 KB "pages". It follows that the flash memory in a node is made up of 32 pages, which we refer to as pages 0 to 31. The firmware is stored in flash memory.

CC2650 also has a separate RF core with an ARM Cortex-M0 processor for communications. It supports a variety of wireless capabilities, including Bluetooth and Zigbee. In addition, the device has a variety of useful peripherals including an AES security module, and several LEDs and buttons. The LEDs and buttons are the main components for user interaction.

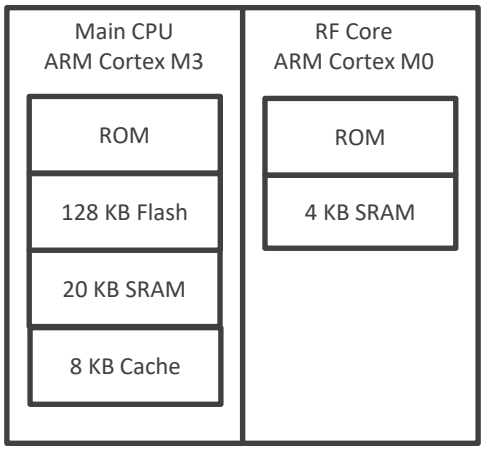

Fig. 3. CC2650 Functional diagram

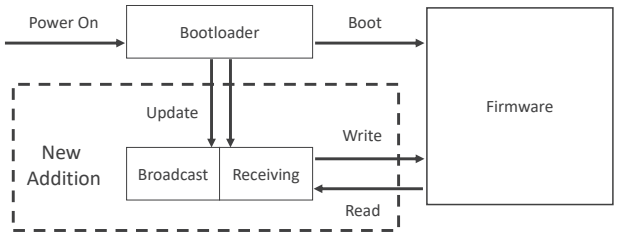

Fig. 4. Booting process

B. Custom Bootloader

The CC2650 already has a bootloader in its read-only memory (ROM) that lets the device boot into the firmware in flash memory. To support OTA firmware update, we created a custom bootloader. The ROM bootloader boots into our custom bootloader first. Then the custom booltloader can either update the firmware or jump to the existing firmware. Figure 4 shows both processes and the custom bootloader is in the dashed box.

The custom bootloader is also stored in flash memory, sharing flash memory with the firmware. Figure 5 shows the layout of our current implementation. Our bootloader requires six pages of flash memory. We place it at the beginning of the flash memory, in pages 0 to 5. In addition, page 31 contains crucial information that the device needs to boot properly. This leaves 25 pages of flash memory for firmware.

To support OTA update, a receiver node must have the custom bootloader installed in pages 0-5 and page 31. The transmitter node also has similar memory layout and a custom bootloader that broadcasts firmware (instead of receiving the firmware).

C. Protocol

Since the firmware is stored in flash memory, a receiver, before writing new firmware to a page in flash, needs to erase

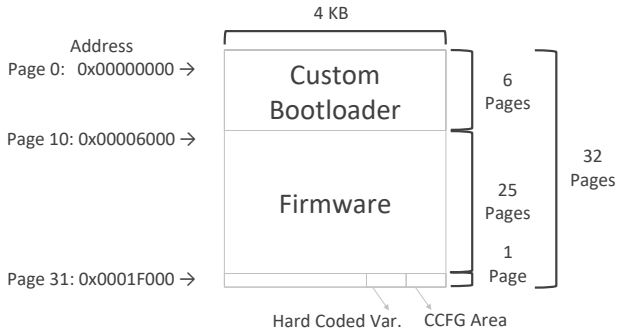

Fig. 5. Flash memory organization

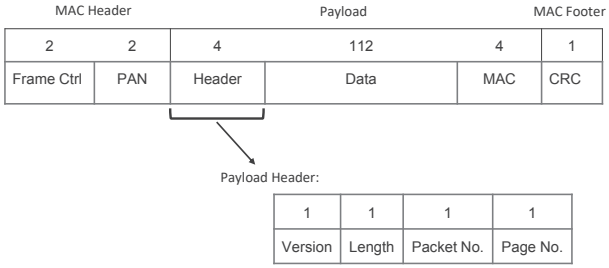

Fig. 6. Data packet structure (in bytes)

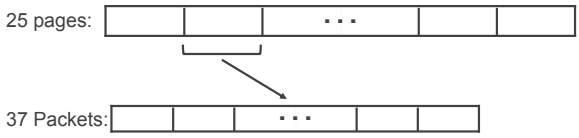

Fig. 7. Firmware divided into packets

all data in the page. Therefore, the transmitter divides the firmware into pages of 4 KB and transmits them one by one.

Per the IEEE standard, the radio module can only transmit 127 bytes at a time. With overhead from physical layer and headers, a packet can carry at most 112 bytes of actual data. Figure 6 shows the structure of a data packet, which follows the IEEE standard. In the figure, MAC header and MAC footer are overhead from the physical layer. The header in the payload contains the length of data in the packet and the information about the location of the packet.

Before sending the data packets for a page, the transmitter sends a header packet containing information such as the total number of pages in the firmware and nonce for encryption.

In our current implementation, the firmware can take up to 25 pages. Each page is further divided into 37 data packets. This is illustrated in Figure 7.

To update many receiver nodes simultaneously, the communication between the transmitter and receivers is one way, from the transmitter to the receivers. The transmitter broadcasts firmware repeatedly. Receiver nodes only accept packets from the transmitter and do not inform the transmitter which packets they have received correctly, or when they are done updating.

The transmitter node broadcasts the new firmware page by page. For each page, it sends out a header packet followed by data packets. It repeatedly transmits the firmware until all the receivers have received the firmware correctly.

To get new firmware, the receiver node boots into the custom bootloader and waits for packets from the transmitter. When a packet is received, the receiver makes sure it is a packet within the network and it has the correct checksum. If a data packet is received correctly, it is written into the flash memory. Before writing the first packet to a page, the receiver erases the page. If a packet is not received correctly, the receiver node will try to receive it in the next round of transmission. Therefore, receivers must keep track of status of all data packets in the firmware. Our implementation uses a collection of bitmaps to track which packets and pages have been successfully received.

In the prototype system we have implemented, a receiver node waits for updates in the custom bootloader and only boots into an existing firmware if a button is pressed. It also uses an LED to indicate that the firmware has been successfully updated. The transmitter is stopped manually when all receivers have the new firmware. In real applications, the receiver can wait for firmware update packets for a period of time. If there is no update request or it already has the most recent firmware, it can boot into the firmware.

D. Security

Regarding security measures, we used the hardware-based AES-CCM (counter with CBC-MAC) module in TI's library [2] for encryption and authentication. A data packet is encrypted before it is transmitted, and is decrypted and authenticated when it is received. The page header packets, as well as the headers in the data packets, are authenticated, but not encrypted as receiver nodes need information for decryption. Currently, all the nodes share the same key, which is predetermined and hardcoded into the custom bootloader.

III. Results

We are interested in the number of transmissions needed for achieving high delivery rate at all receiver nodes, and the time needed for each round of transmission.

A. Firmware Update Success Rate

We tested error rates of transmission in two environments. In one ideal environment, the transmitter and receiver nodes were placed right next to each other, whereas in the non-ideal environment they were placed approximately 16 feet apart with a physical barrier, a wall, between them. Table I shows the packet error rates from a large data set (over 500,000 trials) in the ideal environment. The packet error rate stabilized to 0.04%. This error rate is sufficiently low, so that even if a packet is missing or corrupted in the first round of firmware transmission, it can be received successfully in the second round with very high probability. Table II shows the error rates in the non-ideal environment, which are significantly higher.

TABLE I
ERROR RATES: IDEAL ENVIRONMENT

	Transmitted Packets	Received Packets	Missed Packets	Error Rate
Total Packets	708,780	708,512	268	.04%
Header Packets	20,251	20,248	3	.01%
Data Packets	688,529	688,264	265	.04%

TABLE II
ERROR RATES: NON-IDEAL ENVIRONMENT

	Transmitted Packets	Received Packets	Missed Packets	Error Rate
Total Packets	700,373	629,288	71,085	10.15%
Header Packets	18,431	17,695	736	3.99%
Data Packets	681,942	611,593	70,349	10.32%

It is about 10% for data packets. In both environments the error rate for data packets was significantly higher than the rate for header packets because the header packets are shorter in length.

The probability of successful transmission of the entire firmware is a function of the number of rounds of transmissions M. For simplicity, let us ignore the errors due to header packets and only consider the errors due to the data packets. Let p denote the probability of error for an individual data packet. The probability of successful firmware update is

$$\Pr(\text{success}) = (1 - p^M)^N \quad (1)$$

where N is the number of total data packets. For $p = .04\%$, $N = 925$, the probability of success is 69% when $M = 1$, and improves to 99.99% when $M = 2$. For $p = 10.3\%$, $N = 925$, the probability of success is 90% when $M = 4$, and improves to 99% when $M = 5$.

In practice, the sensor nodes would probably not have to transmit packets through a physical barrier. However, they would be a further distance apart in a location with more noise than the ideal situation. It is likely that the error rates would be somewhere between the rates from the two testing environments.

B. Operational Time

We also measured the timing of the major operations. The CC2650 can write 8 bytes to flash in 8 μs and can erase a page of flash in 8 ms [1]. Writing a packet of 112 bytes takes about 112 μs. Encryption takes $\sim$200 μs per data packet, and decryption takes $\sim$100 μs.

With the addition of encryption and decryption to our protocol, we added a 100 μs wait time before each data packet to ensure enough time for processing on the receiver node.

In addition, we added a 500 μs wait time before each page header packet as accurately and reliably receiving the page header packets is of the utmost importance in our protocol; if a header packet is not received then the receiver will ignore all data packets from that page.

Overall, a firmware of the maximum length 25 pages (102,400 bytes) can be transmitted in about 6.7 seconds. This time is short enough so that relying on retransmission in our protocol is reasonable.

IV. CONCLUSION AND FUTURE WORK

In the project, we studied the over-the-air firmware update problem and provided a solution for a transmitter to update firmware on many receiver nodes simultaneously. We created a custom bootloader that allows a sensor node to either update firmware or boot into an existing one. We designed a protocol for the transmitter to broadcast firmware to receiver nodes. A shared key among sensor nodes is used to encrypt and authenticate data packets.

The current prototype can be improved in many ways, including 1) reducing the size of custom bootloader, 2) using more robust modulation scheme for wireless communications, 3) designing a protocol to propagate new firmware to the entire sensor network, 4) adopting advanced erasure-correction coding schemes to reduce the number of transmission rounds for achieving high delivery rate, and 5) improving the security of the protocol.

REFERENCES

[1] CC26xx Driver Library, Texas Instruments, 2015.

[2] CC2650 SimpleLink™ Multistandard Wireless MCU, Texas Instruments, July 2016.

[3] M. Shavit, A. Gryc, and R. Miucic, *Firmware Update Over The Air (FOTA) for Automotive Industry,* SAE Technical Paper 2007-01-3523, 2007.

[4] M. Hessar, A. Najafi, V. Iyer, and S. Gollakota. *TinySDR: Low-Power SDR Platform for Over-the-Air Programmable IoT Testbeds,* 2019.