Multi-hop Network Re-programming Model for Wireless Sensor Networks

Kangwoo Lee, Jae-eon Kim, Do Thu Thuy, Daeyoung Kim, Sungjin Ahn, Jinyoung Yang
School of Engineering
Information and Communications University
Daejeon, Korea
{kangn2, jekim, dtthuy, kimd, sungjin, jyyang}@icu.ac.kr

Abstract—**Network re-programming in wireless sensor networks is an essential facility, especially in practice, for deploying and maintaining highly dynamic sensor nodes. This paper proposes an efficient way to update whole binary image based on layered architecture with multi-hop network protocol. The layered architecture utilizes 3 different memories including flash, RAM and EEPROM. The network dissemination protocol is based on hierarchical routing and support practical considerations such as sleep/wake-up mechanism for energy saving.**

I. Introduction

Network re-programming is a challenging problem in widely deployed wireless sensor network applications. Usually sensor nodes are located far from existing network infrastructure or easy human accessibility, such as top of trees, side of cliffs, etc. It is very tough and difficult if users want to change sensor network applications on top of existing network. We can say that sensor networks are fundamentally difficult and expensive to maintain.

We propose ANTS (An evolvable Network of Tiny Sensor) PNP(Progressive Network re-Programming) multi-hop network re-programming model for wireless sensor networks to achieve easy and autonomous management for large scale sensor networks. Our network re-programming model can be basically divided into two parts: *PNP Framework* and *Dissemination Protocol*.

Our ANTS platform aims to support an evolvable framework: upgrading kernel module, application, and even kernel core function in runtime without stopping a current application. The proposed PNP framework is based on the layered architecture, taking into consideration how nodes store a lot of packets received during update and how to manage the different types of storages efficiently. The dissemination protocol discusses how to deliver a large program image reliably and quickly. Section II reviews related work. We introduce the PNP framework in section III, and describe the dissemination protocol in section IV. In section V, we evaluate the implementation of ANTS-PNP, and section VI concludes this paper.

II. Related works

A few Network re-programming schemes have been developed, most of which are based on TinyOS. The first implementation of network re-programming in TinyOS is XNP (Crossbow Network Programming) which updates nodes in a single hop range with whole binary image. Deluge is the extension of XNP, which supports multi-hop environment. It is designed to solve the broadcast storm problem by suppressing identical simultaneous broadcast packets. MNP is another variant of XNP-based multi-hop network re-programming protocol, but it proposes a sender selection algorithm for the broadcast storm problem and hidden terminal problem to select the most effective node as a sender. Another XNP-based protocol is MOAP, which uses a sliding window mechanism for reliable data transmission. Incremental network re-programming is the first attempt to transmit only differences between two images for the update by using a simplified variant of Rsync. Aqueduct is a multi-hop network re-programming protocol based on Deluge, support heterogeneous sensor networks by adding a new 'Forward' state to Deluge.These works use a physical addressing which is different from each platform. They stop running application and transit to re-programming state. Moreover, EEPROM is used to store a whole binary image, which is relatively very slow than the flash. ANTS-PNP provides enhanced functionalities such as a relative addressing, simultaneous update, and storage management scheme using the flash than EEPROM.

III. PNP Framework

The PNP framework for ANTS-PNP is one of two key technologies which contributes to the efficiency of our network programming model. In this framework, we propose an approach for a sensor node to store multiple packets while it is in the upgrading phase. Regarding different types of storages (RAM, Flash and EEPROM) have their own characteristics such as storage size, addressing scheme, overhead for accessing, etc.

This research has been supported by the National Computerization Agency and the Ministry of Information and Communication in Korea.

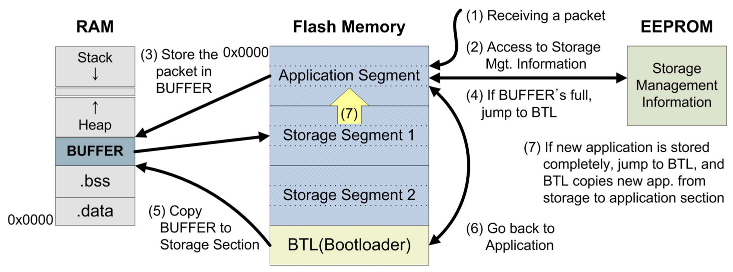

RAM
Flash Memory
EEPROM
Stack
Heap
BUFFER
.bss
.data
0x0000
0x0000
(3) Store the packet in BUFFER
Application Segment
(7)
Storage Segment 1
Storage Segment 2
BTL(Bootloader)
(5) Copy BUFFER to Storage Section
(1) Receiving a packet
(2) Access to Storage Mgt. Information
(4) If BUFFER's full, jump to BTL
(7) If new application is stored completely, jump to BTL, and BTL copies new app. from storage to application section
(6) Go back to Application
Storage Management Information

Fig. 1. Update Procedure of ANTS-PNP

This framework has a management scheme for those storages to best utilize them.

A. *Upgrade procedures*

Figure 1 shows the utilization of different storage in each step of upgrade procedure. Before explaining the procedure, since ANTS-PNP targets whole program image update, rather than modular update, we use the concept "*application*" in this paper to denote whole executable program image, including both the "*real*" application and the operating system. First, (1) a node receives a packet of a new application from a source node. This packet is sent after the update is initiated. (Further details for this step will be described in the next section). Then, (2) the decision of where the packet should be stored is finalized by the storage management information, such as the number of bytes written in a buffer, in EEPROM. Next, (3) the application which is running, stores it in the buffer. (4) If the buffer is full, the execution flow moves to the bootloader (5) to copy the buffer to one of storage sections. To do this, some parameters are also passed (For example, addresses of buffer and storage section). (6) The application continues its execution after writing the buffer. (7) If a new application is received and stored completely, the execution flow moves again to the bootloader, and it copies the new application from the storage section to the application section. Finally, a node restarts with the new, updated application.

B. *Layered Architecture*

ANTS-PNP is based on the layered architecture, as depicted in figure 2, consisting of four layers: *Application Layer, Buffering Layer, Storage Management Layer and Bootloader Layer.* We briefly describe the function of each layer in TABLE I.

TABLE I. FUNCTIONS OF LAYERS IN LAYERED ARCHITECTURE

Layer	Function
Application	Consists of different application versions that exist simultaneously in program memory.
Buffering	Buffers packets before writing to program memory when the full image is received.
Storage Management	Stores necessary information to generate a complete image from received packets and to manage multiple versions of applications.
Bootloader	Stores code to load new version of application to first address of flash or just to execute PIC image.

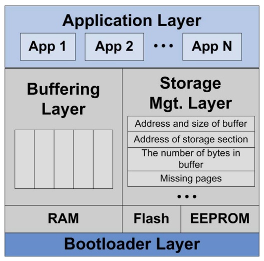


Fig. 2. Layered Architecture of ANTS-PNP

Each layer utilizes different types of storage for quick update and efficient storage usage. An application image can be divided into many pages, which are sent in several packets to fulfill the upgrade procedure.

1) Application Layer: A remarkable difference of PNP framework from other network re-programming schemes is that ANTS-PNP allows multiple applications of different versions to reside simultaeneously in the flash, or program memory. It allows a node to roll back into the previous application, for example, when the new application does not work properly. As the access time to EEPROM is much slower than to flash, we can reduce the update time compared with previous work based on TinyOS which uses EEPROM to store a packet or a bit of application image whenever it is received.

To store different application images, we divide the flash memory into three sections. The size of each section is 40K bytes. Different versions of an applications will be first located in these sections. Whenever there are requests to update to a new version or roll back to a previous version the boot loader will choose the appropriate version from these sections to copy to the application section, which is located at the first address of flash memory.

2) Buffering Layer: Rather than writing a packet whenever received, buffering a certain amount of packets and writing them together allows us to shorten the update time. Since buffering time should be less than writing time to the flash, RAM is used for buffering. For example, Atmega128, one of the most widely used MCUs for sensor nodes, can write 256Bytes to flash in the same time as writing one byte. In addition, the buffer size is the same as that of a page to make it easier to check for missing pages.

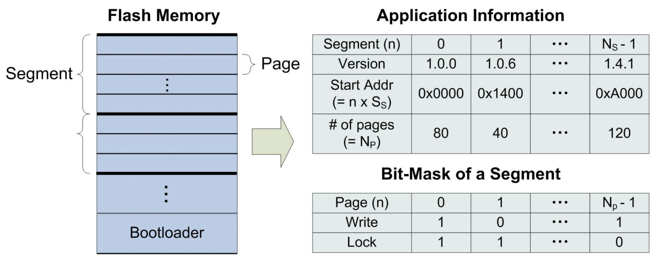
3) Storage Management Layer: Information about storage usage should be kept to generate a complete application image from many packets, and to manage multiple applications. For a complete application image, we need a set of information including: address and size of the buffer, address of the flash to write the buffer, the number of bytes written to the buffer and missing pages. For multiple applications, the information of address, version and length of the applications are required. EEPROM is used to store such information listed above because the information must be preserved even after an update.

4) Bootloader Layer: In order to update a whole application, the current application stops running and the bootloader loads a new application from Application Layer into the first address of the flash. If a PIC (Position Independent Code) can be generated for the platform, the bootloader will simply execute a new application directly without such a loading. Because this is not the case for Atmel, we do not consider PIC at this time, and leave it for further work.

C. Storage Management Scheme

The storage management layer provides storage management functions for the flash and EEPROM such as read/write and page management. Through these functions, the bootloader can write binary images in the flash and verify its correctness. The page management is done by keeping the usage information of pages in the flash in order to determine where a new image should be located.

The approach to manage different storages used in ANTS-PNP is explained in figure 3 below:



Segment (n)	0	1	···	N_S - 1
Version	1.0.0	1.0.6	···	1.4.1
Start Addr (= n x S_S)	0x0000	0x1400	···	0xA000
# of pages (= N_P)	80	40	···	120

Page (n)	0	1	···	N_p - 1
Write	1	0	···	1
Lock	1	1	···	0

Fig. 3. Storage Management Scheme

As shown in figure 3, we propose a new addressing scheme which is independent from the physical address, basing it on two basic units, namely *page* and *segment.* This addressing scheme will also be evolvable with our system, i.e. it can be adapted to multiple platforms such as Atmega128, MSP430, C8051 and usable with modular update scheme. As shown in figure 3, the application information table contains all needed information for updating like version, start address of each segment and number of pages in each segment. For the current approach, the number of pages is fixed, but it can be varied for an evolvable system which requires more flexible memory usage. The Bit-Mask of each segment shows us the status of each segment, whether it is written or not and if it is locked for preventing other applications to change it. We can also estimate the usage of storage using the following equations:

If N denotes the *size of flash for application*, the *size of a section* S_s is: $S_s = N_p * S_p$ where S_p is the size of a page and Np is the number of pages in a section. If E_s is the expected size of a section, N_p is calculated as follows:

$$N_P = \left\lfloor \frac{E_s}{S_P} \right\rfloor$$

In this way, the number of sections, which is calculated by N is:

$$N_S = \left\lfloor \frac{N}{S_S} \right\rfloor$$

IV. Dissemination Protocol

Dissemination protocol for network re-programming has different characteristics/requirements from a typical routing protocol. For example, the size of an application image could be very large compared to usual sensing data. Moreover, a new application must be delivered completely without any error to ensure its proper execution. With some assumptions, listed below, we provide an efficient protocol in a practical situation.

- Update transaction is initiated, at any time, to only interested nodes in a whole network: Usually the user/network manager prepares a new application image and disseminates it from the base node to the whole network.
- Due to the lack of energy resource, of a node, it's important to apply sleep/wake-up mechanism to deployed sensor networks to save energy. Therefore, ANTS-PNP performs network re-programming only to interested node and lets non-interested nodes sleep during the update.
- Because ANTS-PNP is based on a hierarchical routing protocol widely used in practical deployment, a node is assumed to have its parent/child information.

To save the energy for the network, ANTS PNP only choose interested nodes and routing nodes on the way to reach interested nodes using a publish/subscribe mechanism which is represented as ADV-RES. At first, source nodes start 3-way handshake (ADV-RES-SLP) to establish a "*Re-programming Network*" composed of necessary nodes required necessarily to disseminate a new application to interested nodes. Re-programming network contains two types of nodes: one is *interested node* that would be updated; another is *routing node* that is located between interested nodes to forward a new application. Then non-interested nodes sleep during the update. After 3-way handshake, data is disseminated along the re-programming network. While disseminating, two status flags are maintained.

- *Update*: TRUE if a node needs update, FALSE if not.
- *Alive*: TRUE if a node is in the re-programming network, FALSE if not.

The following describes the detail of each step from figure 4.

A. ADV(Advertisement)

A source node floods ADV messages to the whole networks: each recipient forwards the ADV message to child nodes, as

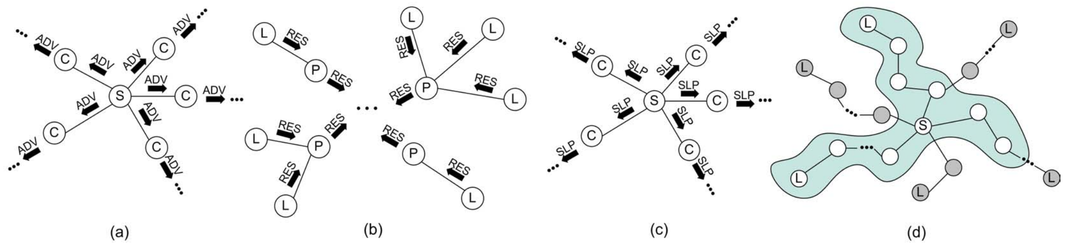


Fig. 4. Dissemination protocol: **(a)** Advertice, **(b)** Response, **(c)** Sleep, **(d)** Re-programming Network

shown in figure (a) of 4. ADV message contains availability and version information of the new application image. Then each node that receives the ADV message determines the version information. TABLE II shows the contents of the version information in a packet.

TABLE II. PACKET OF VERSION INFORMATION

Architecture	Binary Format	Image Type	Image Version
0: AVR 1: 8051 2: MSP430 3. Jennic	0: SREC 1: ihex	0: Whole Kernel + APP 1: Kernel Core 2: Kernel Module 3: Application	Major#.Branch#.Patch#

B. *RES(Response)*

If a leaf node receives an ADV message, it replies, as shown in figure (b) of 4, by sending a RES message including a status of the path which the message goes along: status *Active* is initially FALSE and become TRUE if a node or at least one of its descendents needs the update. Furthermore, *Alive* becomes TRUE if *Active* in received RES message is TRUE, or *Update* is TURE.

C. *SLP(Sleep)*

After receiving RES messages from all child nodes, a source node sends a SLP message to all sensor nodes. The SLP message includes the expected time to finish the update and nodes out of the re-programming network go to sleep for the amount of time after receiving the SLP packet. Figure (c) of 4 shows dissemination of SLP messages, and in (d) white nodes make up the re-programming network. In addition, because time synchronization is critical to synchronize the wake-up time of nodes when the sleep/wake-up mechanism is applied, the time specified in a SLP message is represented in absolute time.

D. *DATA(Data)*

From a source node, a parent node starts sending new application packets by first sending a short *description* packet. A description packet delivers the number of pages and version of the new application. A child node keeps a track of missing packets in the EEPROM by writing only two bytes: one which indicates missing pages 0~255; another is a bitmap of missing packets in that page from 0~7. After receiving the last page, child nodes request retransmission based on the data.

V. IMPLEMENTATION AND TEST

We implement ATNS-PNP on an ANTS-H20 hardware platform. ANTS-H20 is Atmega128 based sensor node capable of RF communication using CC1100 in the frequency of 433MHz. The PNP system architecture is implemented as a module of ANTS-EOS (Evolvable OS), and the dissemination protocol is on top of ANTS-NWK mesh network routing protocol.

We build test environment with one base node as a source node transmitting 15KB image and 5-hop sensor nodes, and measure the number of packets, completion time and retransmission rate, listed in TABLE III. “Total packets” is the number of packets transmitted while updating whole network with 5-hop, including control, data and retransmission packets. “Completion Time” is the time taken from starting update at a source node to finishing update at the last node, and retransmission occurs less than 3 percent of total packets.

TABLE III. AVERAGED RESULT OF MEASUREMENTS

	Total Packets	Completion Time	Retransmission Rate
Measurement	400	42 sec	> 3%

VI. CONCLUSION

ANTS-PNP is an efficient multi-hop network reprogramming model for updating a whole binary image. It’s based on a layered architecture with storage management scheme for the efficient and safe use of the different memories. The PNP framework can be applicable to various platforms and even to evolvable systems. The publish/subscribe-based dissemination protocol provides a practical, efficient update by allowing a sleep/wake-up mechanism during normal functionality and nonstop communication during update.

REFERENCES

[1] D. Kim, T. Sanchez Lopez, S. Yoo, J. Sung, "ANTS platform for Evolvable Wireless Sensor Networks," LNCS, The 2005 IFIP Int’l/ Conf/ on Embedded And Ubiquitous Computing (EUC'2005), Nagasaki, Japan, 6-9 December 2005.

[2] Q. Wang, Y. Zhu, L. Cheng, “Reprogramming Wireless Sensor Networks: Challenges and Approaches,” IEEE Network, May/June, 2006.