효율적인 멀티홉 센서 네트워크 리프로그래밍

이승만° 유민수
한양대학교 정보통신학과
{smlee°, msryu}@rtcc.hanyang.ac.kr

Difference-based Multi-hop Network Reprogramming for Wireless Sensors

Seungman Lee°, Minsoo Ryu
Department of Information and Communications, Hanyang University

요 약

센서 네트워크에서의 네트워크 리프로그래밍은 무선 통신을 통해 센서 노드에서 수행되는 소프트웨어를 변경하는 것으로서, 이를 사용하면 센서 네트워크를 구성하는 수백 개 혹은 수천 개의 노드들의 소프트웨어를 효과적으로 변경시킬 수 있다. 현재 네트워크 리프로그래밍을 위해 XNP, MNP, MOAP, Deluge, Maté 및 Incremental 기법 등이 제안되어 왔지만 아직 만족할만한 방법은 찾아보기 힘든 실정이다. 본 논문에서는 멀티홉을 지원하는 Deluge에 Incremental 기법을 통합하여 효율적인 리프로그래밍 기법을 구현하였다. 제안된 기법은 TinyOS 1.1.14 버전에 구현되었으며, 이를 통해서 네트워크 리프로그래밍을 위해 요구되는 패킷의 수와 소요시간을 크게 단축시킬 수 있다. 또한 센서 노드에서 패킷 전송에 따른 전력소모가 가장 큰 점을 고려할 때 본 연구에서 제안하는 방식을 사용하면 센서 노드의 전력소모량도 크게 감소시킬 수 있는 효과를 기대할 수 있다.

1. 서 론

센서 네트워크에서의 네트워크 리프로그래밍이란 무선 통신을 통해 센서 노드에서 수행되는 소프트웨어를 변경하는 것을 말한다. 일반적으로 센서 소프트웨어를 변경하는 것은 두 가지 방법이 있다. 하나는 센서 노드를 개발용 호스트 컴퓨터에 직접 연결하여 크로스 컴파일 된 코드를 다운로드하는 방법이며, 다른 하나는 무선 통신을 통해 변경된 소프트웨어를 전송하여 센서 노드가 스스로 소프트웨어를 갱신하는 방법이다. 후자의 경우를 네트워크 리프로그래밍이라 하며, 이를 사용하면 수백 개 혹은 수천 개의 노드들로 구성된 센서 네트워크의 소프트웨어를 효과적으로 변경시킬 수 있도록 해준다. 즉 수많은 센서를 사람이 일일이 수거하여 개별적으로 수정할 필요가 없을 뿐만 아니라, 센서 노드를 직접 수거하는 것이 불가능한 상황에서도 소프트웨어의 변경을 가능하게 한다. 대표적인 네트워크 리프로그래밍 방식에는 XNP[2], MNP[3], MOAP[4], Deluge[5], Maté[6]등이 있다.

센서 노드는 일반적으로 제한된 에너지로 구동이 된다. 패킷을 전송하거나 외부 메모리에 접근하는 동작은 다른 동작에 비해서 에너지 소모가 월등히 높다[5]. 또한 무선 네트워크상에서 대량의 패킷을 전달하게 될 때에는 충돌이나 손실이 발생하여 패킷 재전송을 요구하게 되는데, 이때 응용 프로그램의 지연이나 불필요한 에너지 소모를 발생시킨다. 따라서 네트워크 리프로그래밍을 수행하는 과정에서, 전달하는 패킷의 수가 적을수록 그리고 외부 메모리에 대한 접근 횟수가 적을수록 에너지 소모를 줄일 수 있다. 이는 Incremental 기법을 통해서 달성될 수 있으며, 대표적인 Incremental 기법에는 Reijers등[7]이 제안한 방법과 Jeong등[8]이 제안한 방법이 있다. 본 논문에서는 멀티홉을 지원하는 Deluge에 Jeong등이 제안한 Incremental 기법을 통합하여 효율적인 리프로그래밍 기법을 구현하였다. 이를 통해서 네트워크 리프로그래밍을 위해 요구되는 패킷의 수와 소요시간을 크게 단축시킬 수 있다. 또한 센서 노드에서 패킷 전송에 따른 전력소모가 가장 큰 점을 고려할 때 본 연구에서 제안하는 방식을 사용하면 센서 노드의 전력소모량도 크게 감소시킬 수 있는 효과를 기대할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 대표적인 네트워크 리프로그래밍 도구들에 대해서 기술한다. 3장에서는 효율적인 멀티홉 네트워크 리프로그래밍을 설계한다. 4장에서는 실제 구현에 대해서 기술하며, 마지막으로 5장과 6장은 성능평가 및 결론을 기술한다.

2. 관련 연구

센서 네트워크 소프트웨어는 구동 형태에 따라서 플랫폼 고유 시스템(native) 코드 방식과 가상 머신(virtual machine) 코드 방식으로 구분할 수 있다. 그리고 패킷을 네트워크로 전파(dissemination)할 수 있는 범위에 따라서 싱글홉 방식과 멀티홉 방식으로 분류할 수 있다. <표 1>은 대표적인 네트워크 리프로그래밍 도구들을 코드 형식과 전송 범위에 따라 구분해서 보여주고 있다.

여기서 XNP, MNP, MOAP, Deluge 등은 대표적인 네트워크 리프로그래밍 도구이며, Maté는 가상 머신을 사용한 접근방식이다. 특히 XNP, Deluge, 그리고 Maté는 TinyOS[1] 패키지에 내장되어 있으므로 쉽게 응용할 수 있다. 그러나 이들은 Incremental 기법이 적용되지 않았다.

Reijers등이 제안한 방식에서는, 두 프로그램 이미지 사이의 difference를 찾아서 copy, insert, address repair, address patch로 구성된 'edit script'를 생성한

다. 이는 instruction 수준에서 프로그램 코드를 수정할 수 있지만 코드 형식이 프로세서에 종속적이므로 다른 시스템에 적용 시 추가적인 작업이 요구된다. 또한 아직 완벽한 구현이 이루어지지 않았다.

<표 1> 네트워크 리프로그래밍 방식의 비교

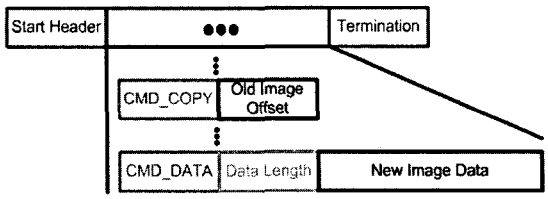
	코드 형식	전파 범위
XNP	native code	싱글홉
MNP	native code	멀티홉
MOAP	native code	멀티홉
Deluge	native code	멀티홉
Maté	virtual machine code	멀티홉
Reijers등	Incremental native (프로세서 종속적)	N/A
Jeong등	Incremental native (프로세서 중립적)	싱글홉 (XNP)
본 논문	Incremental native (프로세서 중립적)	멀티홉 (Deluge)

Jeong등이 제안한 방식에서는, 호스트에서 두 프로그램 이미지 사이의 difference를 찾아서 패킷 형태로 제작한 뒤, 네트워크를 통해 전달하면서 새 버전의 프로그램 이미지를 재구성하는 방식이다. 재구성된 이미지는 센서 노드의 프로그램 메모리 중 부트영역에 존재하는 부트로더에 의해서 응용 프로그램 영역으로 로드되어 실행된다. 이 방식에서는 두 프로그램 이미지를 바이너리 코드 블록 단위로 비교하고 패킷 형태로 전달하므로 프로세서에 중립적인 장점을 갖는다. 하지만 이는 XNP를 기반으로 구현되어 싱글홉으로만 전파되며, 패킷 전달과정에서 새 버전의 이미지를 재구성하게 되므로 전파시간이 길어지게 되어서 에너지 소모 역시 늘어나게 된다.

3. 변경 메시지(Diff-Message) 설계

본 논문에서 제안하는 메시지 전파방식은 변경메시지를 센서 노드용 실행 프로그램 이미지의 형태로 변형 시킨 후, Deluge를 이용해 각 센서 노드에 전달하는 것이다. 이를 위해서 변경 전후의 프로그램 이미지의 비교를 통해 변경메시지(Diff-Message)를 생성하는 방법을 Deluge 호스트에 구현하였으며, 아울러 변경메시지로 실행 이미지를 재구성하고 재부팅하는 기능을 Deluge 센서 노드의 부트로더에 구현하였다. <그림 1>은 변경 메시지의 구조를 보여준다.

변경메시지는 단순하게 그리고 최소의 길이로 설계되며, 메시지 전달과정에서 필요한 체크섬과 같은 오류 검사용 필드는 차후에 실행 파일 형태로 변환하는 과정에서 추가된다. 변경메시지는 두 종류의 명령어를 갖는다. CMD_COPY는 이전 프로그램 이미지의 'Old Image Offset'으로부터 한 블록 크기만큼 버퍼에 복사하라는 명령이고, CMD_DATA는 'Data Length'의 길이만큼 'New Image Data'를 버퍼에 복사하라는 명령이다. 버퍼에는 이전 프로그램 이미지와 메시지로부터 복사된 바이너리 코드만이 존재하므로 실행 이미지가 생성된다.



<그림 1> Diff-Message 구조

4. 구 현

● 하드웨어 구성

기존의 버클리 대학에서 개발한 센서 네트워크 하드웨어인 'MICAZ'에서는 외부 플래시 메모리를 이용하여 네트워크 리프로그래밍을 수행한다. 이때 외부의 플래시 메모리는 Deluge에 의해서 특정한 포맷을 갖게 된다. Deluge 시스템과의 호환성과 구현의 용이함을 위해서 기존의 외부 플래시 메모리를 수정하는 것 보다는 센서 노드에 추가 메모리를 장착하는 방법을 생각해 볼 수 있다. 본 논문에서는 외부에 추가 램을 장착하여 구현하였다. 그러나 대기 전력(Standby Current) 소모를 고려해 볼 때, 외부에 플래시 메모리를 장착하는 것이 더 효과적일 수 있다. 이 추가된 데이터 메모리 영역을 이용하여 새로운 프로그램 이미지를 재구성한다.

● 소프트웨어 구성

Deluge의 구성은 호스트에서 유선으로 싱크노드에게 프로그램 이미지 데이터를 전달하는 Deluge 전용 ①호스트 프로그램과 싱크 노드에서 무선으로 각 센서 노드에게 패킷을 전파하기 위한 ②프로토콜, 그리고 각 센서 노드에서는 전달받은 이미지를 리프로그래밍하고 실행시키는 ③부트로더로 이루어져 있다.

변경된 Deluge ①호스트 프로그램은 변경 메시지 생성 부분과 싱크노드 전달 부분으로 구성된다. 변경 메시지 생성 프로그램은 Rsync 알고리즘[9]을 사용해서 두 프로그램간의 difference를 찾아서 메시지를 생성한다. 생성된 메시지는 먼저 ATmega128[10]에서 실행 가능한 ihex 파일 형태로 가공된다. 다음으로 Deluge 호스트의 싱크노드 전달 프로그램에서 이용 가능한 형태인 xml 파일 형식으로 가공된다. 생성된 변경 메시지는 Deluge의 ②전파 프로토콜의 수정 없이 각 노드까지 전달될 수 있다.

센서 노드의 ③부트로더에는 변경 메시지 디코더와 셀프 프로그래밍(self-programming)[10] 기능을 구현하였다. ATmega128 칩의 부트로더 영역에 존재하는 소프트웨어는 응용 프로그램 영역에 대한 읽기와 쓰기 권한을 갖는다. 이러한 기능을 셀프 프로그래밍이라 하며, 센서 부트로더는 이를 이용해서 자신의 응용 프로그램 소프트웨어를 변경할 수 있다.

5. 성능 평가

본 논문에서 구현한 Incremental 기법의 성능은 원래의 이미지 크기 대 변경 메시지의 크기의 비율을 비교해 봄

으로써 평가될 수 있다. 이를 위해서 다음의 세 가지 경우를 고려하여 실험을 실시하였다. 실험을 위해서 TinyOS에서 제공되는 예제 중 Blink와 OscilloscopeRF를 이용하여 아래와 같이 수정한 후 진행한다. 이때 효율은 다음과 같이 정의한다.

효율(%) = (1 - Diff-Message 크기 / 새 버전의 이미지 크기) * 100

● Case 1: 간단한 파라미터를 수정한 경우의 효율

```
63: command result_t StdControl.start() {
64://Start a repeating timer that fires every 1000ms
65: return call Timer.start(TIMER_REPEAT, 1000);
66: }
```

BlinkM.nc 파일의 65번 라인의 Timer.start(TIMER_REPEAT, 1000)을 Timer.start(TIMER_REPEAT, 500)으로 수정

● Case 2: 기능구현 코드를 추가한 경우의 효율

```
84: event result_t Timer.fired()
85: {
86:     call Leds.redToggle();
87:     return SUCCESS;
88: }
```

BlinkM.nc 파일의 86번 라인 이후에 call Leds.greenToggle(); 을 추가

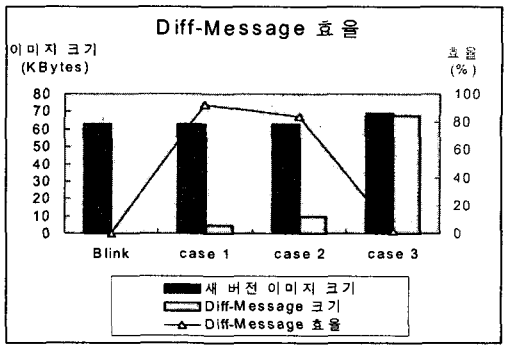
● Case 3: 프로그램을 완전 수정한 경우 이미지 효율

Blink가 실행되는 센서노드를 OscilloscopeRF로 변경 즉, Blink 이미지와 OscilloscopeRF 이미지를 비교하여 변경 메시지 생성

각 결과를 그래프로 나타내면 <그림 2>와 같다. case 1과 case 2의 실험에서 알 수 있듯이, 비교하는 두 프로그램 이미지가 유사한 경우 매우 높은 효율이 있으며, case 3의 경우처럼 두 이미지의 유사도가 현저히 떨어지는 경우라 할지라도 TinyOS의 이미지가 공통으로 포함되므로 여전히 효율이 있다.

6. 결 론

본 논문에서는 센서 소프트웨어를 변경하기위해서 센서 네트워크용 운영체제인 TinyOS에 내재된 네트워크 리프로그래밍 도구인 Deluge에 Incremental 기법을 적용하였다. 이를 통해서 네트워크 리프로그래밍을 위해 요구되는 패킷의 수와 네트워크에서 전파되는 소요시간을 크게 단축시킬 수 있으며, 센서 노드의 전력소모량도 크게 감소시킬 수 있는 효과를 기대할 수 있다.



<그림 2> Diff-Message 효율 실험결과

참 고 문 헌

[1] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. "System architecture directions for networked sensors", In Architectural Support for Programming Languages and Operating Systems, pages 93-104, 2000

[2] Crossbow Technology, "Mote In Network Programming User Reference", TinyOS document, http://webs.cs.berkeley.edu/tos/tinyos-1.x/doc/Xnp.pdf

[3] Sandeep S. Kulkarni, Limin Wang, "MNP: Multihop Network Reprogramming Service for Sensor Networks", Michigan State University, 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05), pages 7-16

[4] Thanos Stathopoulos, John Heidemann and Deborah Estrin, "A Remote Code Update Mechanism for Wireless Sensor Networks", CENS Technical Report # 30, Nov. 2003

[5] Adam Chlipala, Jonathan Hui and Gilman Tolle, "Deluge: Data Dissemination in Multi-Hop Sensor Networks", UC Berkeley CS294-1 Project Report, December 2003

[6] Philip Levis and David Culler, "Maté: A Tiny Virtual Machine for Sensor Networks", ACM ASPLOS, pages 85-95, Oct. 2002

[7] Niels Reijers and Koen Langendoen, "Efficient Code Distribution in Wireless Sensor Networks" WSNA, 2003

[8] J. Jeong and D. Culler, "Incremental network programming for wireless sensors", In Proceedings of the First IEEE International Conference on Sensor and Ad Hoc Communications and Networks, Oct. 2004

[9] Andrew Tridgell, "Efficient Algorithms for Sorting and Synchronization", PhD thesis, Australian National University, 1999

[10] Atmel Corporation ATMega128 Datasheet, http://www.atmel.com