그리드 네트워크를 이용한 대규모 센서 네트워크에서 시스템 재구성 비용을 줄이는 기법

신효정° 차호정
연세대학교 컴퓨터 과학과
{hjshin°, hjcha}@cs.yonsei.ac.kr

Reducing System Reconfiguration Cost for Grid-based Large-scale Wireless Sensor Networks

Hyojeong Shin[0] Hojung Cha
Dept. of Computer Science, Yonsei University

요 약

무선 센서 네트워크 응용 프로그램의 적용 분야가 확대됨에 따라 다양한 응용프로그램을 지원할 수 있도록 응용 및 시스템 기능을 모듈화 하고 네트워크를 통해 업데이트하는 기술이 연구되었다. 센서 시스템의 실시간 코드 교체는 대용량의 코드를 네트워크를 통해 전달해야 하는 고 비용의 동작으로 센서 응용의 자원 절약과 정상적인 응용 수행을 위해 보다 효율적인 관리가 필요하다. 본 논문에서는 모듈화 된 커널 및 응용 코드를 네트워크를 통해 재구성함에 있어 발생하는 오버헤드를 줄이기 위해 네트워크를 영역 별로 나누고 코드의 전파 거리를 최소화하는 코드 전파 기법을 제안한다. 제안된 기법은 대규모 센서 네트워크에서도 예상 가능한 시간 안에 코드 전송을 수행하였다.

1. 서론

센서 네트워크는 실내 또는 실외에 설치되어 무선으로 운용된다. 시스템 개발 및 디버깅의 목적으로 또는 시스템 업그레이드의 목적으로 이미 배포된 코드를 수정해야 할 경우가 자주 일어나게 된다. 보다 효율적인 개발을 위해 노드를 회수하지 않고 네트워크를 통해 코드를 재배포하는 기술은 센서 네트워크 기술 개발의 효용성을 극대화 시켜 준다.

Deluge [1], MNP [2], Trickle [3] 등 동적 코드 전파 알고리즘은 센서 네트워크에 코드 데이터를 복제하여 업데이트 할 수 있도록 연구되었다. 단일 이미지의 복제를 통한 전파에서 시작한 시스템 재구성 알고리즘은 기본적으로 센서 노드들이 같은 바이너리를 설치하는 목적으로 개발되었다. 따라서 다양한 응용 및 시스템 코드에 대해 각각의 노드들이 서로 다른 바이너리를 요구하고 재 설치하는 것이 필요한 경우 사용할 수 없다. 또한 이들 코드 전파 기술은 코드 전파 수행 시간이 길어 수시로 코드 재설치를 해야 하는 네트워크에서는 적합하지 않다. Diff 베이스의 알고리즘 [4][5] 역시 재설치 코드량과 네트워크의 사용량을 줄이긴 하였지만 모든 노드들이 같은 바이너리를 가지고 작동하는 것을 가정 하고 있고, 빈번히 발생하는 코드 전파에 이용하기에 적합하지 않다.

• 본 연구는 한국과학재단에서 지원하는 국가지정연구실사업으로 수행하였음 (과제번호 : 2006-01546)

부분적으로 센서 네트워크를 업데이트 할 경우 코드 업데이트 비용을 줄이는 간단한 방법은 업데이트에 참여하는 노드를 제한하는 것이다. 이것은 업데이트 대상이 되는 코드를 보유한 노드를 가지고 트리 구조를 만들고 싱크 노드로부터 해당 코드를 대상 노드에게 코드를 전달하는 방식이다. 업데이트 트리는 싱크 노드로부터 업데이트 패킷을 네트워크로 Flooding 방식으로 내보내고, 해당 코드를 가진 노드가 이 메시지를 받고 상위 노드로 연결하여 얻을 수 있다. 업데이트 트리를 이용한 노드 제한 방법은 전체 코드 업데이트에 참여하는 노드의 수를 줄여 코드 재배포 비용을 줄일 수 있지만, 다음과 같은 한계를 가진다. 첫째, 네트워크 내에 코드 업데이트에 참여하는 노드의 수가 너무 많을 경우 업데이트 트리를 통한 이득이 없다. 둘째, Flooding 을 통해 트리 빌드 패킷을 전달하므로, 네트워크 사용양이 많으며 대규모 센서 네트워크에 적합하지 않다. 셋째, 싱크 노드로부터 순서대로 코드 업데이트가 진행되어 수행 시간은 센서 네트워크의 크기에 비례하여 커진다. 따라서 대규모 센서 네트워크에서도 효율적으로 부분적인 코드 업데이트를 수행할 수 있는 방법이 필요하다.

본 논문은 배포 이후 센서 네트워크의 응용 프로그램이나 시스템 기능들이 수정되었을 때 발생하는 재배포 비용을 줄이기 위해 Grid-based Code Dissemination System (GDS)기법을 제안한다. 이 기법은 각 노드가 서로 다른 응용 프로그램을 수행하는 환경에서 특정 코드를 업데이트하는 것을 지원한다. 이 경우, 서로 다른 코드를 실행하는 네트워크에서는 모든 노드들이 코드 전파

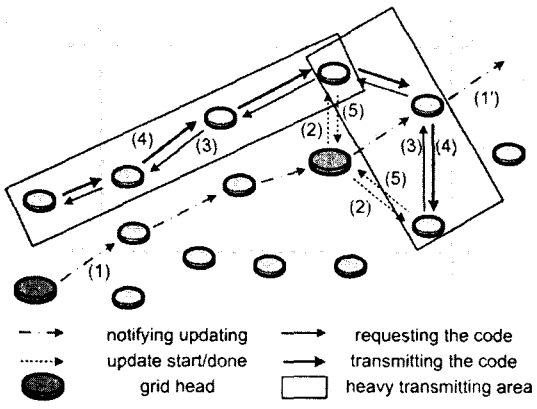

그림 1 그리드 구조를 이용한 코드 업데이트

(1). 하위 그리드 헤드에게 업데이트를 알림
◆ module 을 가지고 있는 node id
(2). 그리드 내에 업데이트가 필요한 노드에게 알림
◆ module 을 가지고 있는 node id
(3). 코드 요청
(4). 코드 전송
(5). 코드 업데이트 완료를 그리드 헤드에게 알림
(6). 첫 업데이트가 성공한 후 하위 그리드 헤드에게 알림 (1'로 점프)
◆ 마지막 설치 한 노드의 id
(7). 헤드노드는 업데이트를 해야 하는 다른 노드에게 알림 (2 로 점프)
◆ 마지막 설치 한 노드의 id
(8). 그리드 내에 모든 노드가 업데이트 된 경우 정지

그림 2 그리드 구조를 이용한 코드 업데이트

에 참여하여 전체 네트워크를 통해 코드 전파를 하는 것은 불필요한 동작이다. 이에 코드 전파 시 해당 모듈을 가지고 있는 노드만 이 업데이트에 참여 시켜 네트워크 부하를 줄이는 코드 전파 방법을 제안한다. 또한 GDS는 코드 전파 시간을 최소화하여 대규모 센서 네트워크에서의 빠른 코드 전파를 제공한다. GDS는 무선 센서 네트워크 용 멀티 스레딩 운영체제인 RETOS [6][7]의 커널 모듈 및 응용 프로그램을 재배포할 수 있도록 설계되었다.

이 논문의 구성은 2장에서 전반적인 시스템의 구조를 설명하고, 3장에서는 제안된 기법의 오버헤드를 분석해 보고 4장에는 결론을 맺는다.

2. 네트워크 분할을 통한 코드 전파 기법

2.1 네트워크 분할 기법

GDS는 센서 네트워크 내의 각 노드가 각각 다른 코드를 수행할 수 있음을 가정한다. 이후 코드의 업데이트가 발생할 경우 전체 네트워크상의 모든 해당 코드를 업데이트해야한다. 이때 센서 네트워크의 크기가 증가하면 코드 업데이트에 드는 비용 또한 증가하게 되며 수행 시간을 예측하기 어려워진다. 제안되는 시스템은 대규모 센서 네트워크의 지원을 위해 전체 네트워크를 분할하여 관리하도록 한다. GDS는 전체 네트워크를 일정 크기로 분할하고 관리하는 그리드 분할 기법을 사용하였다. 각 그리드에는 하나의 헤드 노드가 존재하며 하나의 노드는 각 영역에 존재하는 모든 코드의 위치를 관리하며 헤드 노드끼리 상호 연결되어 전체 네트워크가 유기적으로 연결되어 있도록 한다. TTDD [8] 의 그리드 분할 기법 LEACH [9]의 클러스트 분할 기법 등 지금까지 연구된 여러 네트워크 분할 기법이 모두 사용될 수 있다.

그리드 영역은 해당 영역 내에 있는 모든 코드의 위치를 관리한다. 이 영역이 커지면 영역 내의 노드 수가 많아지고 그리드 헤드가 관리하는 테이블의 크기가 커지고, 관련 메시지 처리로 인한 오버헤드가 커진다. 또한 메모리 크기의 한계로 하나의 헤드 노드에서 관리할 수 있는 정보의 개수는 한계를 가진다.

2.2 코드 전송 및 업데이트

시스템은 그리드 구조를 가지고 있다. 따라서 전체 네트워크에 패킷을 보내기 위해 이미 구성된 그리드 구조를 활용하여 코드 업데이트를 수행하는 것 또한 효율적이다. 헤드 노드는 이미 그리드 내에 코드 분포를 알고 있기 때문에 업데이트에 참여할 노드를 선택 할 수 있다. 그림 1과 2은 센서 네트워크에서 특정 코드의 업데이트 과정을 도식화 한 것이다. 그리드 헤드는 자신의 그리드에 있는 모든 사용 중인 코드를 가진 노드 중 하나의 노드를 업데이트에 참여 시킨다. (1)그리드 헤드는 이웃(상위) 그리드의 헤드로부터 코드 이미지를 가진 노드의 주소를 포함한 code update notification packet 을 전달 받아 업데이트를 시작한다. 헤드는 그리드 내에 업데이트에 참여하는 노드를 순서대로 업데이트 하고 (7)다음 그리드에게 알려주고 종료한다. (2)헤드는 특정 노드를 업데이트시키기 위해 이미지를 가지고 있는 노드의 주소와 함께 코드 업데이트를 지시한다. (3)(4)지시를 받은 노드는 해당 주소로 코드를 요청하고 전달 받는다. (5)업데이트가 끝나면 헤드에게 알려주고 헤드는 (2)다음 노드에게 방금 업데이트 한 노드의 주소와 함께 업데이트 지시 패킷을 보낸다. (6)첫 업데이트가 성공한 경우 하위 그리드에게 알려 하위 그리드도 업데이트 참여할 수 있도록 한다. 이와 같은 방법으로 코드 전파를 수행한다. 코드 전파에 참여한 노드는 그리드 헤드 노드와 업데이트 할 코드를 가진 노드, 그리고 패킷 라우팅 노드들이다.

2.3 전송 수행 시간

기존에 연구된 코드 업데이트는 싱크로부터 전달되는 코드가 전체 센서 네트워크로 차례차례 복사해 나가면서 이뤄진다. 따라서 코드 전파에 드는 시간은 센서 네트워크의 크기와 참여하는 노드의 수에 비례하여 증가하게 된다. 대규모 센서 네트워크라면 이 시간의 비중이 커지

게 되고, 종료 시점을 예측하기 어렵게 된다.

GDS는 그리드의 크기를 통해 코드 전파의 수행 시간을 제한한다. 업데이트 과정은 싱크 노드를 포함한 그리드부터 전체 네트워크의 각 그리드에 하나의 노드만을 업데이트하고, 이후 그리드의 각 노드를 업데이트 하는 두 단계로 이루어진다. 따라서 코드 업데이트 수행 시간은 코드가 전체 그리드에 전달되어 마지막 그리드에 첫 번째 업데이트가 완료되는데 걸리는 시간과 각 그리드의 노드들이 모두 업데이트를 완료하는 시간을 합한 시간이다. 그리드의 크기가 커지면 전체 네트워크에 그리드의 개수가 줄어들어 전체 그리드에 코드를 보내는 데 드는 시간이 줄게 되고, 반면 그리드 내에 존재하는 노드가 많아지게 되므로 노드를 업데이트 하는데 드는 시간이 늘게 된다. 코드 업데이트 수행 시간을 최소화 하는 그리드 크기는 업데이트 되는 코드의 크기 노드의 개수에 따라 달라진다.

그리드로 분할된 네트워크에서의 코드 전파는 코드가 전체네트워크에 우선적으로 전달되고 이후 각 그리드에서 코드 전파를 수행하게 되므로 지역적으로 네트워크 사용이 나눠지는 공간 다중화 (spatial multiplexing) 가 이뤄지게 되어, 네트워크 자원 활용에 효율성을 갖게 되고 보다 빠른 시간에 코드 전파를 가능하게 한다.

3. 분석

코드 전파의 오버헤드를 측정하기 위해 NS-2 [10]를 이용하여 시뮬레이션을 하였다. 표 1 은 Telos Tmote Sky [11] 모트의 성능을 기반으로 한 시뮬레이션의 설정 값이다. 노드의 플래시와 메모리 중 기본적으로 설치된 시스템과 응용 프로그램 수행에 사용되는 양을 제하고 그 운영에 영향을 주지 않는 범위에서 코드 전파 응용 프로그램에 메모리를 할당하였다. 코드 전파 메커니즘은 각 노드의 시스템 휴지 시간에 수행되며 여유분의 메모리를 사용하며 설치 된 운영체제 및 실행 중인 응용 프로그램과 독립적으로 수행된다. 일반적으로 센서 노드의 메모리는 제한적이며 실행중인 응용 프로그램과 운영체제에 의해 얼마간 사용되어 코드 전파 메커니즘을 위해 할당되는 메모리는 제한이 있다. 그리드 헤드는 그리드 내의 모든 코드의 위치 정보를 관리하여야 하는 데 메모리 한계를 극복하기 위해 플래시 메모리 공간을 함께 할당하여 자주 사용되지 않는 데이터는 플래시 공간에 저장하여 그 용량을 크게 늘릴 수 있었다. Telos Tmote sky와 Mica 시리즈의 하드웨어를 기준으로 헤드 노드는 RAM 공간에 약 500개, 플래시 공간에 약 1500개를 관리할 수 있다. 따라서 헤드 노드는 약 2000개의 코드의 위치 정보를 관리할 수 있다. 하나의 노드가 10개의 응용 프로그램을 수행하고 있다면 하나의 그리드에는 200개 이상의 노드를 포함할 수 없다는 제약을 갖는다. 시뮬레이션에서는 그리드 크기가 12 hop 일때, 그리드 내에 노드 200개를 배포하였다. 그리드는 현실적으로 10 hop 이하로 제한되어야 한다.

코드 업데이트 시뮬레이션은 이미 사용하고 있는 특정 코드를 수정하고 재 배포하는 실험이다. 새로 수정된

표 1. 시뮬레이션 세팅

파라미터	기본 값
cache size of head node (RAM)	5 kBytes
cache size of head node (Flash)	18 kBytes
code size	1.5 kBytes
radio range (1 hop)	10 m
node placement	random

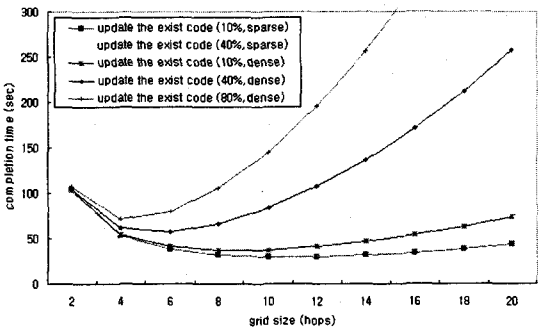

그림 3 그리드 크기에 따른 업데이트 수행 시간

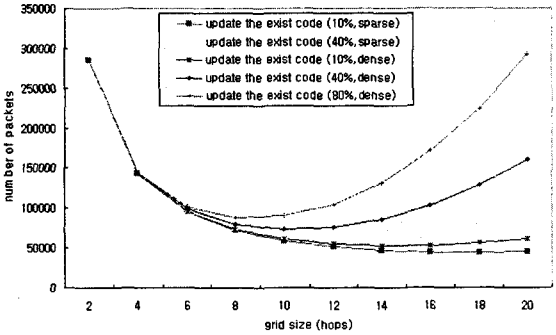

그림 4 그리드 크기에 따른 패킷 사용량

코드를 전체 네트워크에 투입하고 전체 네트워크에 반영되는 시각까지의 시간과 네트워크 사용량 및 사용된 전력량을 측정하였다. 전체 네트워크 크기는 100 홉 * 100 홉 이며, 그리드 크기를 다양하게 바꿔 가며 시뮬레이션 하였다. NS-2 시뮬레이션은 메모리 및 수행 시간의 제약으로 대규모 센서 네트워크를 그래도 반영하여 시뮬레이션 할 수 없다. 따라서 실험을 용이하게 하기 위해 패킷 전송을 하지 않는 노드를 제거하고 센서 네트워크를 분할하여 시뮬레이션 하고 통합하는 등의 방법을 사용하여 시뮬레이션하였다.

새로운 코드를 전체 네트워크에 배포하거나 이미 배포 된 코드를 수정하는 것은 오버헤드가 크다. 네트워크의 크기가 커질수록 그 코스트는 지수 함수로 증가하게 된다. GDS는 전체 네트워크를 그리드 단위로 분리하고 코드를 전파하기 때문에 시간과 네트워크 사용량에서 이득을 가진다. 그림 3 과 4는 100 hop * 100 hop 의 대규

모 센서 네트워크에서 기존의 코드를 새로 업데이트 하는 경우 그룹의 크기에 따른 수행 시간과 코스트를 보여준다. 각 라인은 네트워크에 분포 된 노드의 빈도와 업데이트되어야할 코드의 빈도수에 따라 다른 비용을 의미한다. 괄호 속 비율은 노드 중 해당 코드를 가진 노드의 비율이며, 고밀도 네트워크는 100 평방미터 당 6개의 노드를, 저밀도 네트워크는 3개를 배포하였다. 코드 추가의 경우 대규모 네트워크를 그리드 단위로 구분하였기 때문에 하나의 코드는 우선 그리드 단위로 전파 된다. 그리드 크기가 커지면, 네트워크 내에 그리드의 수가 줄어들게 되어, 코드를 전체 네트워크에 전달하는 시간이 줄어들지만, 그리드 크기가 너무 커지면 그리드 내에서 수행되는 코드 업데이트 시간이 증가하여 전체 수행시간은 다시 늘어나게 된다. 코드 업데이트는 추가로 그리드 내에 존재하는 모든 코드에 대해 코드 업데이트를 수행하여야 하므로, 그리드 내의 사용되는 코드의 수가 많을 수 록, 노드의 빈도가 높을 수 록 이에 따른 오버헤드가 커지게 된다.

4. 결론

본 논문은 동적으로 센서 시스템을 수정하면서 발생하는 오버헤드를 줄이기 위해, 센서 네트워크를 분할 관리하고 코드 분포를 관리하여 코드 재전송 시간을 줄이는 코드 전파 기법을 제안하였다. 제안된 기법은 기존의 코드 전파 기법 보다 적은 자원을 사용하여 코드 전파를 하였으며, 대규모 센서 네트워크에서도 예상 가능한 시간 안에 모든 수행을 마쳤다.

앞으로의 센서 네트워크는 실제 배포 및 사용에 있어 동적이고 융통성이 있는 작업 수행을 요구하게 될 것이다. 같은 센서 노드도 별도의 설정 작업 없이 배포되는 장소에 따라 스스로 환경에 맞는 코드를 갖추고 수행하는 기능은 대규모 센서 네트워크의 배포를 용이하게 할 것이며, 이것은 제안되는 시스템을 통해 실현될 수 있다. 우리는 적응형 센서 운영체제 구축의 일환으로 배포 된 센서 네트워크의 코드 설치 내역을 관리하고 원격지에서 코드를 설치, 제거할 수 있는 툴을 개발하고 있다. 코드 전파 기법 또한 커널 모듈 및 응용 프로그램 단위의 재설치가 가능한 RETOS 시스템에서 사용할 수 있도록 개발하고 있다. 또한, 작업 환경에 따라 노드의 역할 및 구성 요소를 스스로 결정하고 구축하는 시스템을 연구하고 있다.

참고 문헌

[1] Jonathan W. Hui and David Culler, "The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale," *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, Baltimore, MaryLand, USA, November 2004.

[2] S.S.Kulkarni and Limin Wang, "MNP: Multihop Network Reprogramming Service for Sensor Networks," *Proceedings of the 25th International Conference onDistributed Computing Systems (ICDCS'05)*, Columbus, Ohio, USA, June 2005.

[3] Philip Levis, Neil Patel, Scott Shenker and David Culler, "Trickle: A Self-Regulating Algorithm for Code Propagation and maintenance in Wireless Sensor Network," *Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI'04)*, San Francisco, California, USA, March 2004.

[4] Jaein Jeong and David Culler, "Incremental network programming for wireless sensors," Proceedings of the first IEEE Communications Society Conference onSensor and Ad Hoc Communications and Networks (IEEE SECON'04), Santa Clara, California, USA, October 2004.

[5] Jeol Koshy and Raju Pandey, "Remote incremental linking for energy-efficient reprogramming of sensor networks," *Proceedings of the Second European Workshop on Wireless Sensor Networks (EWSN'05)*, Istanbul, Turkey, January 2005.

[6] Hyoseung Kim and Hojung Cha, "Towards a Resilient Operating System for Wireless Sensor Networks," Proceedings of The 2006 USENIX Annual Technical Conference (USENIX'06), Boston, MA, June 2006.

[7] Sukwon Choi and Hojung Cha, "Application-Centric Networking Framework for Wireless Sensor Nodes," *Proceedings of the 3rd Annual International Conference on Mobile and Ubiquitous Systems: Networks and Services (MOBIQUITOUS'06)*, San Jose, California, USA, July 2006.

[8] Haiyun Luo, Fan Ye, Jerry Cheng, Songwu Lu and Lixia Zhang, "TTDD: A Two-tier Data Dissemination Model for Large-scale Wireless Sensor Networks," *Proceedings of the 8th ACM International Conference on Mobile Computing and Networking (MobiCom'02), Atlanta, Georgia, USA, September 2002.*

[9] Wendi Rabiner Heinzelman, Anantha Chandrakasan and Hari Balakrishnam, "Energy-efficient communication protocol for wireless microsensor networks," *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences (HICSS'00)*, Maui, Hawaii, USA, January 2000.

[10] Kevin Fall and Kannan Varadhan, editors. *ns* notes and documentation. The VINT Project, UC Berkeley, LBL, USC/ISI, and Xerox PARC, November 1997. Available from http://www-mash.cs.berkeley.edu/ns/.

[11] Telos Tmote Sky, http://www.moteiv.com.