Multi-round Rsync 알고리즘을 이용한 에너지 효율적인 센서 네트워크 리프로그래밍 기법

구원모° 박용진
한양대학교
{wmku°, park}@hyuee.hanyang.ac.kr

Energy-Efficient Reprogramming of Sensor Networks using Multi-round Rsync Algorithm

Wonmo Ku° YongJin Park
Hanyang Univ.

요 약

본 논문에서는 TinyOS 기반의 센서 네트워크에 대한 리프로그래밍을 에너지 효율적으로 수행하기 위한 매커니즘을 제안한다. 베이스 스테이션에서 센서노드에게 프로그램 전체를 보내는 대신 이전 버전과의 차이인 델타를 생성해서 전송할 때 Multi-round Rsync 알고리즘을 적용해 델타 파일의 크기를 최대한 줄이는 기법과 업데이트가 불필요한 플래시메모리 페이지에 대한 업데이트를 방지하기 위한 페이지 맵 기법을 통해 Rsync만을 사용하는 기존 방식보다 최대 30% 이상 에너지를 절감할 수 있음을 확인하였다.

1. 서 론

센서 네트워크는 그동안 군사분야, 환경분야, 보안 및 재난방지 등의 특화된 분야에서 중요한 역할을 수행해 왔으나 유비쿼터스 시대를 맞아 앞으로 광범위한 분야에 적용이 예상되고 다양한 센서 네트워크 응용 프로그램이 등장할 것이다. 센서 네트워크 응용 프로그램은 특성상 적게는 수십 개에서 많게는 수백에서 수천 개의 센서노드에 탑재되어 동작 하므로 응용 프로그램 개발과정에서나 버그수정, 기능개선 등으로 이미 설치된 센서 네트워크를 리프로그래밍 해야 할 경우 이들 센서노드를 효율적으로 리프로그래밍하는 방법이 필요하다. 특히 센서노드는 한정된 자원으로 운용되므로 효율적인 리프로그래밍 방법은 센서노드의 에너지를 되도록 적게 소모하도록 설계할 필요가 있다.

센서 네트워크를 리프로그래밍하기 위한 방법으로 노드에 직접 케이블을 연결해 프로그램을 다운로드하는 ISP(In-System Programming)이 있다. 이것은 센서노드의 개수가 증가 할수록 작업시간이 많이 소요되며 사람 손이 닿지 않는 곳에 설치된 노드에 대한 리프로그래밍을 어렵게 만들고 무엇보다 작업과정을 자동화할 수 없는 단점이 있다. 이에 반해 네트워크 프로그래밍[1]은 베이스 스테이션이 무선링크를 통해 업그레이드 할 프로그램을 센서노드들에 전송하는 방식으로 센서 네트워크 프로그래밍 작업을 자동화할 수 있게 해주고 전체 네트워크 프로그래밍에 소요되는 시간도 줄여 주는 이점이 있어 효율적인 네트워크 프로그래밍 방안에 대해 더욱 활발한 연구가 필요하다.

TinyOS 1.1[2] 릴리즈에서는 네트워크 프로그래밍 기능을 지원하기 위해 XNP (Crossbow Network Programming)[1]모듈이 구현되어 있다. 이 모듈은 프로그램 전체를 센서노드들에게 전송하는 방식이여서 네트워크 프로그래밍에 소요되는 시간과 에너지가 많이 소모되는 단점이 있다.

Incremental Network Programming for Wireless Sensors(이하 INP)[3]에서는 XNP 방식의 단점을 개선하여 네트워크 프로그래밍 시간을 줄이고자 Rsync 알고리즘[4]을 이용해서 이전 버전과 새 버전의 차이(delta)만을 효율적으로 생성하고 델타를 XNP 커맨드 스크립트 형태로 전송하는 방식을 제안하였다. INP가 XNP에 비해 네트워크 프로그래밍 시간측면에서 향상을 보이기는 했으나 델타를 생성하는 과정과 새 버전을 센서노드 프로그램 플래시메모리로 리프로그래밍하는 과정에서 여전히 개선할 점이 있다.

본 논문에서는 전송해야할 델타의 크기를 INP 방식보다 더 줄일 수 있는 방안과 새 버전을 프로그램 플래시메모리에 리프로그래밍할 때 에너지를 절감할 수 있는 방안을 제안한다. 본 논문의 구성은 다음과 같다. 2장에서는 네트워크 프로그래밍에 관한 배경지식을 설명한다. 3장에서는 INP 방식을 개선한 INPMR (Incremental Network Programming using Multi-round Rsync Algorithm) 매커니즘을 소개한다. 4장에서는 네트워크 프로그래밍에 소비되는 전류소모량 측면에서 INP 대비 INPMR의 성능을 비교하고 이후 결론을 맺는다.

2. 배경지식

2.1 XNP

네트워크 프로그래밍은 보통 3단계를 거쳐 수행 된다: (1) Encoding (2) Dissemination (3) Decoding.[3] Encoding 단계에서 베이스 스테이션은 프로그램 이미지를 쪼개어 무선링크로 전송할 수 있는 TinyOS AM

(Active Message) 패킷 포맷으로 만든다. Dissemination 단계에서 각 패킷은 무선링크를 통해 센서노드로 전송되고 이를 수신한 센서노드는 외부 플래시메모리에 수신한 패킷을 SREC 포맷[5]으로 저장한다. Decoding 단계에서 센서노드의 네트워크 프로그래밍 모듈은 수신한 프로그램 이미지를 검사한 후, 부트로더를 호출해 내부 프로그램 플래시메모리로 리프로그래밍한 후 센서노드를 리셋 시킨다. 이후 센서노드는 바뀐 프로그램으로 동작하게 된다.

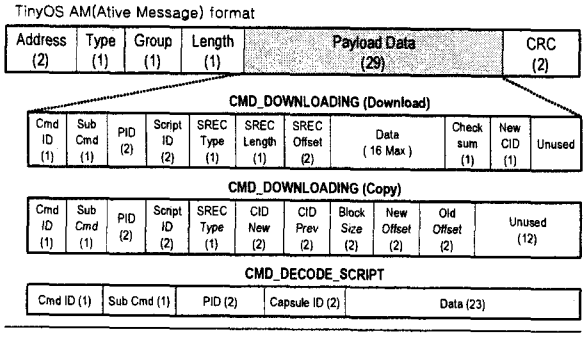
2.2 INP

2.2.1 베이스 스테이션 동작

원래의 Rsync 알고리즘을 센서 네트워크 상황에 맞게 고쳐 에너지 소모가 많은 체크섬 리스트 생성 작업은 베이스 스테이션이 수행하도록 수정하여 적용하였다. 베이스 스테이션은 변형된 Rsync 알고리즘을 이용해서 이전 버전에 대한 델타를 생성해 내고 이를 커맨드 스크립트 형태로 센서노드로 전송한다.(그림 1) 새 버전 영역 중에서 이전 버전과 같은 영역에 대해서는 copy 커맨드 스크립트를 생성하고 다른 영역에 대해서는 download 커맨드 스크립트를 생성한다. TinyOS AM 패킷 하나 당 XNP 커맨드 스크립트 하나를 실어 보낸다.

2.2.2 센서노드 동작

센서노드는 수신된 커맨드 스크립트 메시지를 외부 플래시메모리에 저장한다. 스크립트 메시지를 빠짐없이 모두 수신하고 나면 베이스 스테이션에서 디코드 커맨드를 보낸다. 이 커맨드에 의해 저장해 놓은 스크립트 메시지와 이전 버전을 이용해서 새 버전을 만들고 부트로더를 호출한다. INP에서는 새 버전이 저장되어 있는 메모리 번지가 매번 달라지므로 부트로더를 호출할 때, 새 버전의 시작번지를 함수인자로 지정해 주는 것을 제외하고 나머지 동작은 XNP와 동일하다.



(그림 1) INP 커맨드 스크립트 메시지 구조

3. INPMR (Incremental Network Programming using Multi-round Rsync Algorithm)

INP 방식은 XNP에 비해 베이스 스테이션에서 전송해야 될 스크립트 메시지 개수를 줄여 전체 네트워크 프로그래밍에 소요되는 시간을 줄이기는 했지만 네트워크 프로그래밍과정에서 소모되는 에너지측면에서는 관심을 두지 않았다. 네트워크 프로그래밍 진행 중에 센서노드에는 스크립트 메시지의 저장과 새 버전 생성 그리고 내부 프로그램 메모리로의 리프로그래밍 과정이 발생하는데 이때 플래시메모리 쓰기 동작이 빈번히 일어난다. 이는 <표 1>에서 보는 바와 같이 센서노드 동작 중 전류 소모가 가장 많은 부분이다. INP 방식보다 플래시메모리 쓰기 동작을 줄이면서 네트워크 프로그래밍이 가능하다면 프로그래밍에 걸리는 시간뿐만 아니라 에너지 절감 측면에서 큰 이점이 있을 것이다. 이 장에서는 네트워크 프로그래밍을 수행하는 데 있어 플래시메모리에 대한 액세스 횟수를 줄이기 위한 목적으로 INP를 개선한 INPMR 방식을 제안한다.

출처: [6]

Operation	nAh
Receive a Packet	8.000
Transmit a Packet	20.000
Flash Read Page	1.111
Flash Erase/Write Page	83.333

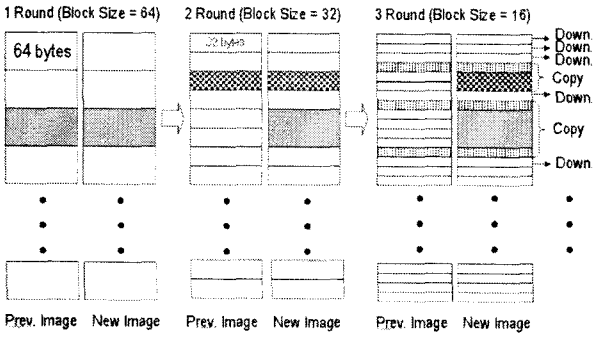
<표 1> Mica[7] 계열 센서노드 동작별 전류소모량

3.1 Multi-round Rsync 메커니즘

전송해야 될 스크립트 메시지 개수를 줄이기 위해 Multi-round Rsync[8] 알고리즘을 이용하였다. 이 방식의 핵심은 블록크기를 달리하면서 Rsync 알고리즘을 여러 번 수행하는 것이다. 본 논문에서는 각 라운드마다 블록크기를 이전 라운드 블록크기의 1/2로 줄이면서 Rsync 알고리즘을 반복해서 INP에서는 찾지 못했던 여러 크기의 블록을 찾고자 하였다

INP에서는 블록 크기를 64바이트로 설정하고 Rsync 알고리즘을 적용해 스크립트 메시지를 생성하였다. INP에서 사용한 JarSync[9] 뿐만 아니라 오프소스에서 제공하는 MD4 해시함수는 최소한 64바이트 블록을 입력받도록 구현되어 있어 64바이트 보다 더 작은 블록은 찾을 수 없는 단점이 있다. INPMR에서는 64바이트 보다 더 작은 블록들도 찾기 위해 32바이트나 16바이트 블록 비교에서는 MD4 해시함수를 사용하지 않고 strcmp()함수를 이용해서 데이터 스트링을 직접 비교하는 방식을 사용하도록 코드를 수정하였다.

(그림 2) INPMR의 동작방식을 보면, 매 라운드 마다 이전 라운드에서 찾은 블록들은 제외하고 나머지 영역에 대해 Rsync 알고리즘을 적용한다. 최종 라운드가 끝나면 각 라운드에서 찾은 여러 블록들 중 인접한 블록들은 하나로 통합된다. 이렇게 통합된 블록에 대해 copy 스크립트 메시지 를 생성하고 나머지 영역은 download 메시지를 생성한다. (그림 2)에서 첫 번째 그림이 INP의 결과이고 마지막 그림이 INPMR로 스크립트를 만드는 경우이다. INPMR은 더 작은 블록들 여러 개를 찾고 인접한 블록들은 하나로 통합해서 스크립트를 생성하므로 이 예제에서 INP에서 5개의 download 스크립트로 보내야 할 영역이 INPMR에서는 copy 스크립트 한 개로 해결된다.



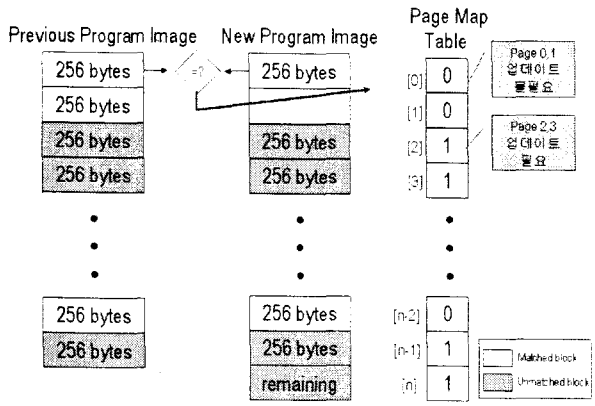
(그림 2) Multi-round Rsync 예제

3.2 페이지 맵

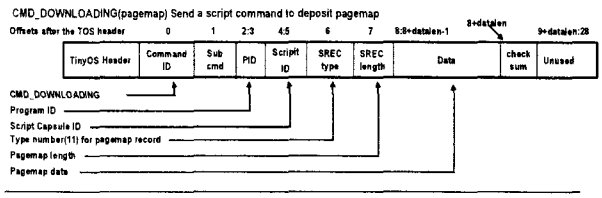
페이지 맵 기법은 외부 플래시메모리에 생성된 새 버전을 내부 프로그램 메모리로 리프로그래밍하는 단계에서 소모되는 에너지를 절약하기 위해 적용한다. 기존 XNP나 INP의 부트로더는 외부 플래시메모리에 저장되어 있는 새 버전 전체를 내부 프로그램 메모리로 한 페이지 단위씩 리프로그래밍하도록 설계되었다. 만약 업데이트 하고자 하는 페이지의 내용이 이전 버전과 동일하다면 이 페이지 영역은 굳이 업데이트 할 필요가 없으므로 페이지별 업데이트 여부를 알려 주는 정보가 있다면 프로그래밍을 효율적으로 할 수 있을 것이다.

3.2.1 베이스 스테이션 동작

이전 버전과 새 버전을 페이지 크기인 256 바이트 블록들로 나누고 차례로 비교하면서 페이지 맵을 생성한다. 업데이트가 필요한 페이지는 '1'로 표시한다 (그림 3). 생성된 페이지 맵을 페이지 맵 커맨드 스크립트 메시지 형태로 만들어 센서노드로 전송하는데 페이지 맵 스크립트 메시지는 copy나 download와 같은 다른 스크립트 메시지보다 먼저 보낸다 (그림 4). 페이지 맵 커맨드 스크립트 메시지는 XNP나 INP에 없던 메시지 형태이므로 INPMR에서 새로 추가하였다.



(그림 3) 페이지맵 생성 예제



(그림 4) 페이지맵 커맨드 스크립트 메시지

3.2.1 센서노드 동작

페이지 맵 메시지를 처리하여 램 영역에 페이지 맵 테이블을 만든다. 부트로더는 내부 프로그램 메모리로 리프로그래밍하는 단계에서 이 테이블을 이용하여 불필요한 페이지 업데이트를 방지한다.

4. 성능 평가

4.1 실험절차

INP에서 실시한 테스트 시나리오<표 2>를 그대로 사용해 INP방식과 정확한 비교가 되도록 하였다. 각 방식에서 생성되는 스크립트 메시지 개수에 <표 1>,<표 3>의 파라미터 값을 적용해 네트워크 프로그래밍에 소요되는 총 전류량을 구한 후 이를 비교하였다. INPMR에서는 추가적으로 페이지 맵을 생성해 센서노드로 전송하므로 페이지 맵을 전송하는데 필요한 스크립트 메시지 개수도 총 스크립트 메시지 개수에 포함했으며 페이지 맵을 통해 불필요한 페이지 업데이트를 줄임으로써 얻게 되는 이득도 총 전류량 계산에 반영하였다.

	Original Prog.	New Prog.	Change
Case 1	XnpBlink	XnpBlink	constants changed
Case 2	XnpCount	XnpCount	a few lines added
Case 3	XnpBlink	XnpCount	major change
Case 4	XnpCount	XnpCount	IntToLeds module removal
Case 5	XnpCount	XnpCount	IntToRfm module removal

<표 2> 테스트 시나리오

4.2 전류소모량 계산식

센서노드가 소모하는 총 전류량을 정밀하게 계산하기 위해 네트워크 프로그래밍 과정을 4단계로 세분화하고 <표 1>,<표 3>의 파라미터를 적용하여 각 단계별 전류소모량 계산식을 구하였다.<표 4>

- 1단계: 베이스 스테이션에서 전송하는 모든 스크립트 메시지를 수신하는데 필요한 전류소모량을 구하는 과정이다. 총 스크립트 메시지 개수에 패킷수신 시 전류소모량을 곱하였다
- 2단계: 수신된 메시지를 외부 플래시메모리에 저장하는데 필요한 전류소모량을 구하는 과정이다. 메시지는 외부 플래시메모리에 저장될 때 32바이트 단위로 저장되므로 메시지 개수에서 32바이트를 곱

Parameter	Description
N_d	download 커맨드 스크립트 메시지 개수
N_c	copy 커맨드 스크립트 메시지 개수
N_m	페이지 맵 전송에 필요한 메시지 개수
N_{srec}	SREC 포맷으로 생성되는 새 버전 이미지의 SREC 라인 개수
N_{fs}	SREC 포맷으로 생성되는 새 버전 이미지의 실제 프로그램의 바이트 단위 크기
N_{ps}	플래시메모리 페이지 크기: 256 바이트
N_{samepg}	새 버전 페이지들 중에서 이전 버전과 주소와 내용이 모두 같은 페이지의 개수

<표 3> 성능측정에 필요한 Parameter

해서 실제 플래시메모리에 저장되는 총 바이트 수를 구하고 이를 플래시메모리 페이지 단위로 변환한 후 페이지 쓰기 동작에 필요한 전류소모량을 곱하였다.

- 3단계: 플래시메모리에 저장되어 있는 커맨드 스크립트 메시지를 처리해서 새 버전을 생성하는데 필요한 전류소모량을 계산한다. copy 커맨드 스크립트 메시지로 이전 버전의 내용을 복사해 오고 download 커맨드 스크립트 메시지는 메시지에 포함된 데이터를 사용해서 외부 플래시메모리 빈 공간에 새 버전을 만든다. 이 단계가 빈번한 플래시메모리 액세스로 인해 전류소모량이 가장 많다. 이 과정은 커맨드 스크립트 메시지를 모두 읽는데 필요한 전류소모량과 새 버전의 크기를 SREC 포맷으로 플래시메모리에 저장하는데 필요한 전류소모량의 합으로 계산하였다. 새 버전은 SREC 포맷으로 플래시메모리에 저장되기 때문에 저장해야 할 총 바이트 크기는 N_{srec}에 32를 곱해준 값이 된다.
- 4단계: 새 버전을 내부 프로그램 플래시메모리 영역으로 프로그래밍 하는데 필요한 전류소모량을 계산한다. 부트로더는 한번에 한 페이지를 읽은 후 SREC 라인 단위(32바이트)씩 처리한다. SREC 한 라인에는 실제 프로그램 코드 16바이트가 들어 있어 16 바이트씩 프로그램 플래시메모리 내부 버퍼영역으로 옮기는데 내부 버퍼에 256바이트가 모여지게 되면 플래시메모리 한 페이지를 업데이트 한다. XNP나 INP에서는 새 버전을 통째로 내부 프로그램 메모리로 리프로그래밍하지만 INPMR에서는 페이지 맵을 이용해서 업데이트가 필요한 페이지에 대해서만 리프로그래밍하므로 이전 버전과 같은 내용을 갖는 페이지의 개수를 구한 다음 이 값은 페이지 업데이트 횟수에서 제외시킨다.

4.3 실험결과

<표 5>,<표 6>,<그림 5>에서 보는 바와 같이 INPMR 방식이 INP방식보다 네크워크 프로그래밍에 소요되는 전류소모량이 더 적은 것으로 확인되었다. 이 실험은 Multi-round Rsync 기법이 다양한 크기의 이전 버전과 같은 블록들을 찾아 스크립트 메시지 개수를 줄일 수 있

단계		단계별 전류소모량 계산식
1단계 I_{step1}	INP	$(N_d+N_c)\cdot 8.0$ nAh
	INPMR	$(N_d+N_c+N_m)\cdot 8.0$ nAh
2단계 I_{step2}	INP	$((((N_d+N_c)\cdot 32)+N_{ps}-1)/N_{ps})\cdot 83.3$ nAh
	INPMR	$((((N_d+N_c+N_m)\cdot 32)+N_{ps}-1)/N_{ps})\cdot 83.3$ nAh
3단계 I_{step3}	INP	$(((N_d+N_c)\cdot 32+N_{ps}-1)/N_{ps})\cdot 1.1$ nAh + $(((N_{srec}\cdot 32)+N_{ps}-1)/N_{ps})\cdot 83.3$ nAh
	INPMR	$(((N_d+N_c+N_m)\cdot 32+N_{ps}-1)/N_{ps})\cdot 1.1$ nAh + $(((N_{srec}\cdot 32)+N_{ps}-1)/N_{ps})\cdot 83.3$ nAh
4단계 I_{step4}	INP	$(((N_{srec}\cdot 32)+N_{ps}-1)/N_{ps})\cdot 1.11$ nAh + $((N_{fs}+N_{ps}-1)/N_{ps})\cdot 83.3$ nAh
	INPMR	$(((N_{srec}\cdot 32)+N_{ps}-1)/N_{ps})\cdot 1.11$ nAh + $(((N_{fs}+N_{ps}-1)/N_{ps})-N_{samepg})\cdot 83.3$ nAh
총 전류소모량 $I_{programming}$		$I_{step1} + I_{step2} + I_{step3} + I_{step4}$

<표 4> 단계별 전류소모량 계산식

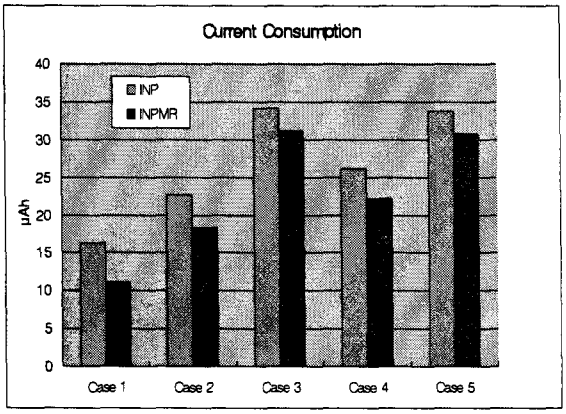
다는 것과 페이지 맵을 활용하면 불필요한 업데이트 동작을 방지함으로써 전체 전류소모량을 줄일 수 있다는 사실을 보여주었다. <표 5>에서 변경사항이 적은 경우에는 생성된 바이너리 파일 구조가 이전 버전과 유사하여 내용이 같은 페이지 개수가 많고 변경사항이 많아질수록 페이지 개수가 적어져 페이지 맵은 변경사항이 적은 경우에 효과가 큼을 확인할 수 있다.

	Case 1	Case 2	Case 3	Case 4	Case 5
FileSize	43.6KB	44.8KB	44.8KB	44.3KB	44.3KB
N_{srec}	1017	1045	1045	1034	1033
N_{fs}	16224	16670	16584	16494	16486
INP $N_d + N_c$	8	322	945	525	933
INPMR $N_d + N_c$	6	191	810	336	798
N_m	1	1	1	1	1
N_{samepg} / Total pages	62/64	20/65	3/65	6/65	3/65

<표 5> 스크립트 메시지 개수

	Case 1	Case 2	Case 3	Case 4	Case 5
INP $I_{programming}$	16.3μAh	22.6μAh	34.1μAh	26.2μAh	33.7μAh
INPMR $I_{programming}$	10.9μAh	18.3μAh	31.1μAh	22.1μAh	30.8μAh
Current Save	33.1%	19.0%	8.8%	15.6%	8.6%

<표 6> 네크워크 프로그래밍 전류소모량



(그림 5) 전류소모량 비교

5. 결론

본 연구에서는 네트워크 프로그래밍에 소요되는 전류소모량을 절감하기 위해 Multi-round Rsync 알고리즘과 페이지 맵 기법을 적용한 INPMR 방식을 제안하였고 이 방식은 기존 INP방식 대비 전류소모량 측면에서 전반적으로 우수한 성능을 보여 주었다. 이전 버전 대비 수정사항이 적은 경우에는 33% 이상, 수정사항이 많은 경우에도 8% 이상의 에너지를 절감하였다. INPMR은 INP보다 생성되는 스크립트 메시지 개수가 적고 불필요한 페이지 업데이트를 방지하므로 네트워크 프로그래밍에 소요되는 시간측면에서도 INP보다 우수한 성능을 예상할 수 있고 본 논문에 제시된 기법은 XNP 방식을 기반으로 Mica 계열 센서노드에서 동작하도록 설계되어 TinyOS 기반의 센서 네트워크 분야에 즉시 활용될 수 있는 것 또한 장점이다.

참 고 문 헌

[1] Jaein Jeong, Sukun Kim and Alan Broad, "Network Reprogramming", TinyOS document, http://www.tinyos.net/tinyos-1.x/doc/NetworkReprogramming.pdf

[2] TinyOS : http://www.tinyos.net

[3] Jeong, J., Culler,D.:Incremental network programming for wireless sensors. In First IEEE Comm. Soc. Conf. on Sensor and Ad Hoc Communications and Networks. 2004

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

[5] Motorola S-record format : http://www.amelek.gda.pl/avr/uisp/srecord.htm

[6] Alan Mainwaring et al,. Wireless Sensor Networks for Habitat Monitoring. WSNA '02

[7] Crossbow Technology Inc. *Mica Motes* http://www.xbow.com

[8] J. Langford. Multiround rsync. January 2001. Unpublishedmanuscript. http://www-2.cs.cmu.edu/~edu/~jcl/research/mrsync/mrsync.ps

[9] Casey Marshall, "Jarsync: a Java implementation of the rsync algorithm," http://jarsync.sourceforge.net/