센서 네트워크 임베디드 소프트웨어 기술동향

2004. 12
목차

Ⅰ. 서 론 ................................................................. 73

Ⅱ. 기술 개발 동향 .................................................. 74
   1. 개요 ................................................................. 74
   2. 연구개발 동향 .................................................. 75
      가. TinyOS ......................................................... 75
      나. TRON ........................................................... 78
      다. 저전력 멀티-홉(multi-hop) 네트워크 기술 .................... 93
      라. 국내 기술 개발 현황 ......................................... 97

Ⅲ. 기술 특성 분석 .................................................... 98

Ⅳ. 결론 및 전망 ..................................................... 99

참고자료 ............................................................. 101
센서 네트워크 임베디드 소프트웨어 기술동향

I. 서론

센서 네트워크 기술은 새로운 컴퓨팅 패러다임인 유비쿼터스 컴퓨팅의 전입 기술로 중요한 의미를 갖고 이런 새로운 패러다임의 기술이 성공적으로 발전하기 위해서는 기반이 되는 전입기술의 확산이 중요하다. 본문에서는 센서 네트워크를 위한 임베디드 소프트웨어 기술의 동향과 오픈소스 프로젝트로 진행 중인 TinyOS 내부 구조를 분석하고 센서 네트워크의 특성과 TinyOS에 구현된 기능을 분석한다. TinyOS는 새롭게 떠오르고 있는 센서 네트워크에 대한 개념 정립과 실제 구현을 동시에 진행하고, 그 결과를 공개하여 센서 네트워크의 확산을 이끌고 있다. 실제 서비스를 진행한 예가 있기 때문에 가장 앞서나가는 센서 네트워크 기술이라 할 수 있다. 현재까지 개발된 TinyOS 기술을 중심으로 센서 네트워크 임베디드 소프트웨어의 주요 요소를 분석하여 지금까지의 센서 네트워크 기술개발 상황과 향후 산업화 가능성과 파악한다.
Ⅱ. 기술 개발 동향

1. 개요

센서 네트워크(sensor network)는 물리공간의 상태인 빛, 소리, 온도, 움직임 같은 물리적 데이터를 센서노드에서 감지하고 측정하여 중앙의 기본 노드(base-station or sink)로 전달하는, 센서 노드들로 구성되는 네트워크이다. 센서 네트워크는 일반적으로 멀티-호(multi-hop) 무선 네트워크 형태의 다수의 분산 센서 노드들로 구성된다. 센서 노드들은 하나이상의 센서(온도, 소리, 빛, 가속도, 자기장 등), 액추에이터(actuator), 마이크로컨트롤러, 수십 KB 크기의 EEPROM, 수 KB의 SRAM, 수백 KB 크기의 플래시 메모리, 근거리 무선 통신 모듈로 구성된다. 센서 네트워크 기술은 셀서와 무선 네트워크 기능을 이용하여 물리공간에서 측정한 아날로그 데이터를 디지털 신호로 변환하고, 인터넷 같은 전자 공간에 연결된 루트(root) 노드로 전달하는 입력시스템의 역할을 한다. 물리적 세계와 사이버 세계를 연결할 수 있는 특징 때문에, 센서 네트워크의 개념은 새롭게 대두되고 있는 지능형 서비스들, 지능형 환경 모니터링, 위치인식 서비스, 지능형 의료시스템, 지능형 모봇 시스템 등 다양한 분야에 적용되고 있다.

센서 네트워크의 장점은 낮은 사양의 하드웨어를 이용하여 무선 에드-호(ad-hoc) 네트워크를 구성할 수 있는 점이다. 예를 들어, 지금까지 개발된 블루투스(blue tooth), 무선랜(wireless LAN) 등의 무선 네트워크 기술들은 반드시 컴퓨터, PDA 같은 고급
센서 네트워크 임베디드 소프트웨어 기술동향 75

컴퓨팅 장치를 필요로 하는데, 센서 네트워크 노드는 독자적으로 네트워크를 구성한다. 이런 네트워크 구성의 용이성 때문에 유비 퀘티스(ubiquitous) 컴퓨팅 환경의 기반기술로 사용될 수 있을 것이다. 인텔도 기술 로드맵 중에 센서 네트워크를 기반 기술로 예측하고 있다(1).

근래 센서 네트워크와 관련된 칩과 시스템들이 상용화되기 시작했으나 널리 보급되고 산업화되기 위해서는 해결되어야 할 문제들이 많이 남아있다. 네트워크 프로토콜, 적절한 응용 서비스 등의 부재로 인해 빠른 시간 내에 상용화되기 힘들 것으로 예측되기도 한다.

다음 장에서 분석할 TinyOS는 새로운 컴퓨팅 패러다임인 저가의 소형 컴퓨터 시스템, 즉 네트워크 임베디드 시스템 또는 센서 네트워크에 사용될 수 있는 시스템 기술이다. 버클리 대학에서 주도적으로 개발을 추진하고 있으며, 현재 센서 네트워크 분야에서 실제 구현 및 적용에 가장 빠르게 대응하고 있다. 인텔 및 크로스보우 등의 회사가 이 기술의 확산을 위해 지속적으로 투자하고 있으며 마이크로소프트사도 관련 기술을 보유하기 위해 노력 중이다.

TinyOS의 표면적인 목표는 지그비 표준과 다르게 보일 수도 있지만, 주요 요소기술들은 동일한 기능을 지향하고 있다. 이런 이유로, 근래 본격적인 구현 및 산업화가 시도되면서 두 기술에 대한 업계의 관심이 높아지고 있다.

TinyOS의 장점은 누구에게나 공개되어 사용할 수 있다는 점인데, 기존의 공개 소프트웨어와 동일한 장점을 갖는다. 리눅스
유비쿼터스1) 나 비에스디 와 비슷한 기술 공개의 원칙은 산업화 및 기술 확산을 가속화 시킬 것으로 예측된다.

2. 연구개발 동향

가. TinyOS

TinyOS의 개발 목적은 세 가지로 설명할 수 있다2). 첫째, 미래의 센서 네트워크 노드에 사용될 수 있는 소프트웨어 구조를 설계, 둘째, 여러 하드웨어 플랫폼과 소프트웨어 라이브러리를 사용하여 운영체제와 서비스 애플리케이션을 구현할 수 있도록 설계, 셋째, 제한된 자원, 동시성, 강건성, 응용 서비스별 요구사항을 만족할 수 있는 설계이다.

이런 목표의 달성을 위해 TinyOS는 구성이 모듈화 되었으며, 제한된 자원에서 동작할 수 있는 이벤트 구동 방식으로 구현되었다. 모듈구성에 따라 운영체제가 다양한 하드웨어에 적용이 가능하며, 서비스 애플리케이션도 동일한 추상화를 유지할 수 있다.

센서 네트워크 노드들은 무선 통신 데이터들을 실시간으로 처리해야 하기 때문에 높은 동시성이 지원되어야 한다. 각 센서 네트워크 노드들은 제한된 저장용량의 상황에서 여러 작업을 동시에 처리해야 하며 강건한 시스템의 특성을 유지하기 위해, 이벤트 기반의 동시성을 지원해야 한다.

일반적인 컴퓨팅 장치들과 달리 대부분의 센서 네트워크 노드들은 유지의 입력을 필요로 하지 않기 때문에, TinyOS에서 센
서 노드를 동작시키는 것은 하드웨어 이벤트와 태스크로 볼 수 있다. 하드웨어 이벤트는 인터럽트를 의미하며, 이런 인터럽트는 타이머, 센서, 통신 장치로부터 발생된다. 태스크들은 일정시간의 지연을 갖는 프로시저 호출을 의미하고, 하드웨어 이벤트 또는 다른 태스크를 호출할 수 있다. 태스크는 큐에 등록된 후 실행되는 데, 태스크들이 처리되는 동안에도 하드웨어 인터럽트가 실행될 수 있으며 높은 우선순위를 갖는다. 태스크 큐에 실행되어야 할 태스크가 존재하지 않을 경우, 시스템은 새로운 하드웨어 인터럽트가 발생될 때까지, 슬립 모드로 전환된다. 태스크 큐를 운영하는 방법은 태스크 호출되면 태스크 큐에 해당 태스크를 등록하고 태스크가 실행 될 때 큐에서 해당 태스크를 삭제한다. 태스크 실행 후 태스크 큐에 더 이상 실행할 태스크가 남아 있지 않으면 바로 슬립(sleep) 모드로 전환된다.

TinyOS는 컴포넌트 단위로 시스템 모듈화를 구성한다. 컴포넌트는 몇몇 상태와 태스크로 구성된다. 각 컴포넌트는 다른 컴포넌트들과 인터페이스를 통해 호출을 통해 동작을 연결시키고 인터페이스는 커맨드와 이벤트로 분류된다. 인터페이스를 통해 컴포넌트는 외부와 연결된다. 커맨드는 어떤 동작에 대한 호출이며 이벤트는 호출된 어떤 동작이나 환경변화에 의해 실행되는 것을 의미한다. 이벤트의 리턴 값으로는 처음 호출한 동작의 성공 여부를 알 수 있다. 특정 이벤트들은 하드웨어 인터럽트에 직접 연결되어 동작한다. 서비스 애플리케이션 개발자들은 컴포넌트들을 라이브러리로 사용하며, 인터페이스들을 사용하여 컴포넌트들을 서로 연결하여 애플리케이션을 만든다.
TinyOS의 동시성 모델은 불로킹 같은 기법을 지원하지 않기 때문에, 결과적으로 많은 동작 호출들이 스플릿-페이즈(split-phase)로 동작하게 된다. 어떤 동작에 대한 호출이 커맨드를 통해 실행되고, 이에 대한 성공여부를 이벤트를 통해 회신한다. 일반적인 반응을 필요로 하는 함수나 하드웨어의 제어에는 이 방법이 적당하지만, 상위 애플리케이션 개발을 복잡하게 만드는 단점이 있다. 순차적인 동작에 따라 애플리케이션을 개발할 때, 경우에 따라 상태-머신(state-machine) 기법을 사용해야만 하는 경우가 발생한다. 이런 이유 때문에 매우 제한된 앱 자원 환경에서 하나의 스택으로 동작이 가능하고 오류처리가 정확해 진다.

TinyOS에서는 시스템과 사용자 부분을 나누어 놓지 않았다. 그러나 대부분의 애플리케이션에서 자주 사용되는 모듈들이 있는데, 타이머, 데이터 수집, 전력 제어, 네트워크 모듈들이 대표적인 예이다. 이 모듈들의 사용은 센서 네트워크 노드가 한정된 전력으로 동작하고 주기적으로 데이터를 수집하며, 간단한 데이터 처리와 주 위 노드로 데이터를 전송한다는 주요 동작에 대한 특징을 나타내고 있으며, 다른 센서 네트워크 플랫폼인 벤티스(MANTIS)$^3$, 센서심(SENSORSIM)$^4$, 엠휘타(EMSTAR)$^5$등도 동일한 특징을 갖고 있다.

(1) 센서 네트워크 프로그래밍 언어 nesC$^6$

TinyOS는 애플리케이션 개발을 위해 프로그래밍 언어로 nesC를 사용한다. nesC는 컴포넌트 기반이며 문법은 C 프로그래밍 언어와 유사하다. nesC는 일반의 프리-프로세서(pre-processor) 방식으로 nesC 컴파일러는 소스코드를 C 프로그램 파일로 변환하
고, 이 파일은 GCC 컴파일러로 컴파일과 링크를 담당한다.

nesC 프로그래밍은 컴포넌트(component)들로 구성되며 컨피규레이션(configuration)과 모듈(module)로 세분화 된다. 컨피규레이션은 컴포넌트들의 연결(wiring)을 담당하고 모듈은 메시지 핸들링, 시그널 프로세싱 같은 특정 서비스를 제공한다. 컴포넌트가 플랫폼의 세부 기능을 담당하는 부분이라 할 수 있고, 예를 들면 시스템 타이머를 구현한 부분이 Timer 컴포넌트로 구현될 수 있다. 복잡한 기능의 컴포넌트인 경우에는 여러 컴포넌트가 서로 연결될 수도 있다.

nesC 컴포넌트는 커맨드(command)와 시그널(signal)을 기반으로 동작한다. 커맨드는 C 프로그래밍 언어의 함수와 같은 기능이며, 페러미터(parameter) 전달, 호출에 대한 리턴(return), 동작 명령을 수행하는 단위이다. 시그널은 이벤트가 발생했을 때 실행되는 함수이며, 일반적으로는 하드웨어 인터럽트와 연결된 함수이다. 커맨드와 반대로 상위 애플리케이션 방향으로 명령을 수행한다. TinyOS의 응용 애플리케이션은 이런 컴포넌트들이 서로 연결된 형태이다.

nesC의 인터페이스(interface)는 서로 다른 컴포넌트들을 연결하기 위한 인터페이스를 제공한다. 컴포넌트를 구현할 때는 실제
동작들에 각각에 대해 인터페이스를 정의해 주어야 하며, 다른
컴포넌트에서 이 기능을 사용할 때에는 인터페이스에 연결한다.

```c
#include <clock.h> // clock.h defines constants

interface clock{
    // Configure the clock event interval
    command result_t setRate(uint8_t interval, uint8_t scale);

    // Clock interval has elapsed
    async event result_t fire();
}
// end of interface
```

nesC는 센서 네트워크를 위한 새로운 방식의 프로그래밍 언어로써 기존의 C 프로그래밍 언어에 비해 다양한 라이브러리의 재사용이 가능한 장점이 있고 실제 코드의 작성 길이를 줄일 수 있다.

<그림 4> 컨피규레이션(configuration)
<그림 5> 모듈(module)

```c
module LedsC{
    provides{
        interface Leds;
    }
}

implementation Leds{
    uint8_t ledsOn;
    async command result_t Leds.redOn() {
        atomic {
            TOSH_CLR_RED_LED_PIN();
            ledsOn |= RED_BIT;
        }
        return SUCCESS;
    }
}
```

으며, 하드웨어 인터럽트 처리를 컴포넌트의 시그널을 이용하여 쉽게 처리할 수 있는 장점이 있다.

(2) TinyOS 애플리케이션 구조

TinyOS는 일반적인 컴퓨터 운영체제와 달리 운영체제와 애플리케이션이 분리되어 설치되지 않는다. <그림 6>와 같이 운영체제 부분과 응용 애플리케이션 부분이 함께 컴파일 되어 하드웨어 플랫폼에 다운로드 된다. 그림에서 Application 부분을 사용자가 직접 만들게 되며, 나머지 부분은 기존 라이브러리를 재사용할 수 있다.

이런 구조에서는 메인(main) 함수가 사용자에게 직접 보이지 않는다. <그림 7>에서 Main 컴포넌트가 사용되고 있다. 이 컴포넌트의 구현 부분이 메인 함수를 포함하고 있다. Main 컴포넌트는
플랫폼의 초기화, 사용되는 모듈의 기능 호출, 테스크 스케줄러의 실행을 담당한다. 기능의 실행은 컨피규레이션에 있는 컴포넌트들의 실제 구현 부분에서 확인할 수 있다. <그림 7>에서 Main. StdControl에 연결되어 있는 부분들이 실제 동작을 위해 사용되는 컴포넌트들이다. BlinkTask의 기능은 1초에 1번씩 붉은색 LED가 점등하는 것이고 실제 구현부분에는 BlinkTaskM, Single Timer, LedsC가 사용되고 있다.

메인 함수는 리얼메인(RealMain.nc) 컴포넌트에서 호출된다. 메인 함수에서는 먼저 hardwareInit을 호출하여 하드웨어 초기화를 하고, Pot.init으로 무선 통신에 사용할 세기를 결정한다. TOSH_sched_init로 테스크 스케줄러를 초기화 시키고, StdControl.init와 StdControl.start를 통해 사용자가 생성한 코드를 초기화한 후 실행된다. 마지막으로 Interrupt.enable로 인터럽트 사용 가능으로 설정하고 테스크 스케줄러를 무한 루프로 실행 시작한다.

<그림 6> TinyOS 애플리케이션 구조
(3) 테스크 스케줄러

센서 네트워크 환경에서 각 노드가 동작하도록 할 수 있는 이벤트는 하드웨어 인터럽트이다. 센서 노드는 초기화 후 하드웨어 인터럽트의 발생에 따라 동작한다. TinyOS에서 지원하는 프로세스(process)는 두 가지로 구분할 수 있다.

이벤트(event)는 하드웨어 인터럽트에 의해 발생하는 것으로 대표적으로는 타이머, 센서에 의해 발생한다. 실시간성을 요구하는 데이터가 발생하는 것이기 때문에 가장 우선순위가 높으며 짧은 시간동안 동작을 한다.

테스크(task)는 태스크-탑 컴퓨터에서 실행되는 쓰레드(thread)와 비슷한 개념으로 센서 노드에서 함수 실행에 동시성을 부여하는 것이다. 실시간으로 처리되지 않으며 테스크 스케줄러에 의해 순차적으로 실행된다. 또한 실행 중에 발생하는 이벤트에 의해서 실행이 지연 된다.

센서 노드는 대부분 슬립(sleep) 상태에 있고, 실제 동작은 한정된 짧은 시간에 이루어진다. 센서 노드들은 전원이 충분하지 않은 상태이기 때문에 전력 소모를 줄이기 위해 최대한 슬립모드로 동작 시킨다.

TinyOS 테스크 스케줄러는 단순하게 구현되어 있다. 테스크 큐가 존재하고 코드 실행 중 테스크가 호출될 때에 해당 함수의 주소를 큐에 입력한다. 테스크의 실행은 바로 이루어지지 않고 진행 중인 루틴이 모두 끝난 후에 테스크 큐에 존재하는 테스크들을 차례로 호출한다. 하드웨어 인터럽트는 최우선 순위를 갖는다.
액플리케이션에서 태스크를 호출하는 방법은 함수를 구현할 때 task라는 타입을 정해 주고, 호출할 때 post를 사용하여 호출한다. <그림 9>는 태스크 호출 방법을 나타낸다. 이 그림의 ADC.data Ready(uint16_t data) 함수가 호출될 때, 동작 순서는 putdata(data)가 실행되고 post processData에 의해 processData 함수의 포인터(pointer)가 태스크 큐에 입력되고, 곧이어 바로 return SUCCESS가 실행된다. 태스크 큐에 입력된 processData는 모든 루틴이 종료된 후에 실행된다. <그림 10>의 TOS_post(thread_pointer)가 post processData()와 연결되는 함수이다. 여기서 확
인할 수 있듯이 post 호출은 함수를 실행시키지 않고 함수의 포인터만을 테스크 큐에 입력시킨다.

테스크 스케줄러는 테스크 큐에 남아있는 함수를 계속 실행시키고, 테스크 큐에 실행할 함수가 남아있지 않으면 일정시간 동안 슬립상태가 되고, 이후에 웨이트(wait) 모드로 상태가 변경된다. 만일 하드웨어 인터럽트가 전혀 발생하지 않는 상태가 지속된다.

<그림 8> TinyOS 테스크 스케줄러

![TinyOS Kernel Scheduler Diagram]
멘 센서 노드는 계속해서 슬립과 웨이트 상태를 주기적으로 반복할 것이다. 이런 경우 전력 소모는 최소화 될 것이다.

TinyOS 스키줄리의 특성이 있는데, 스키줄리의 운영 원리는 각 태스크가 반드시 종료된다는 것을 가정하고 있다. 만약 태스크가 무한 루프를 실행하며 종료되지 않는다면 센서 노드는 더 이상 다른 작업을 실행하지 못한다. 이는 스키줄리가 무한 루프를 강제로 종료할 수 있는 기능이 없기 때문이다.

단일 애플리케이션만을 관리할 수 있도록 설계되어 있어서 기존 운영체제보다 기능이 간략화 되어 있다. 앞으로 하드웨어가 더 발달되고 복잡한 데이터 처리 또는 GUI(graphic user Interface) 등 많은 사용자 인터페이스를 지원해야 되다면 TinyOS 커널은 다양한 기능을 지원해야 할 것이다.

<그림 9> 태스크 호출

```c
SenseTaskM.kc // ADC data ready event handler
async event result_t ADC.dataReady(uint16_t data) {
    putdata(data);
    post processData();
    return SUCCESS;
}

task void processData() {
    int16_t i, sum=0;

    atomic {
        for (i=0; i < size; i++)
            sum += (rdata[i] >> 7);
    }
    display(sum >> log2size);
}
```
/*
 * TOS_post (thread_pointer)
 * 
 * Put the task pointer into the next free slot.
 * Return 1 if successful, 0 if there is no free slot.
 * 
 * This function uses a critical section to protect TOSH_sched_free,
 * as tasks can be posted in both interrupt and non-interrupt context,
 * this is necessary.
 */

bool TOS_post(void (*tp)()) __attribute__((spontaneous))
{
__nesc_atomic_t fInterruptFlags;
uint8_t tmp;

fInterruptFlags = __nesc_atomic_start();
tmp = TOSH_sched_free;

if (TOSH_queue[tmp].tp == NULL)
{
    TOSH_sched_free = (tmp + 1) & TOSH_TASK_BITMASK;
    TOSH_queue[tmp].tp = tp;
    __nesc_atomic_end(fInterruptFlags);
    return TRUE;
}
else {
    __nesc_atomic_end(fInterruptFlags);
    return FALSE;
}
}

void TOSH_run_task() {
    while (TOSH_run_next_task())
    {
        TOSH_sleep();
        TOSH_wait();
    }
}

(4) 하드웨어 플랫폼

TinyOS를 지원하는 플랫폼은 “MOTE”가 있다. MOTE는 Tiny
OS가 처음 발표된 이후로 다양한 형태로 발전되어 왔다. <표 1>
에 지금까지 개발된 MOTE의 종류를 정리 하였다.
이 다양한 MOTE들은 동일한 마이크로컨트롤러 제품군(Atmel AVR)을 사용하지만, 무선 통신 모듈이나 인터페이스 등의 다른 주요 부분이 다르다. TinyOS가 사용될 수 있는 하드웨어 중에는 인텔에서 개발한 ARM 프로세서와 블루투스 통신 모듈을 이용하는 하드웨어 플랫폼이 있고, 유럽에서도 블루투스를 이용한 센서 네트워크 플랫폼을 개발했다.

근래에는 노르웨이의 칩콘(ChipCon)에서 2.4GHz 대역의 통신 칩이 판매 시작되어, 이 통신 칩을 이용한 센서 네트워크 플랫폼들이 개발되어 판매가 시작되고 있다. 칩콘에서는 미국에 Moteiv사를 설립하여 텔로스(TELOS) 플랫폼을 개발하였다(그림 11). Mica와 Mica2를 사용한 제품인 MicaZ도 개발하였다(그림 12).

TinyOS의 최신 버전에는 IEEE 802.15.4 MAC 프로토콜이 지원되며, 앞에 언급한 두 가지 2.4GHz 하드웨어 플랫폼에 사용될 수 있다. 지난 6월 버클리 대학이 산타크루즈(Santa Cruz)에서 개최
한 NEST retreat\textsuperscript{10}에서는 TinyOS를 이용한 텔로스와 MicaZ그리고 CC2420DBK\textsuperscript{7} 플랫폼간의 통신 시연이 있었다.

TinyOS는 IEEE 802.15.4 규격을 지원하기 시작했고 지속적으로 구현이 진행되고 있다. IEEE 802.15.4는 지그비(Zigbee)규격에서도 요구하는 링크, 물리 프로토콜 규격이다. TinyOS도 지그비와 같은 규격인 IEEE 802.15.4를 지원하게 되어, 무선 통신이 호환 가능하다. 지그비 제품을 만들기 위해 TinyOS 플랫폼을 사용할 수 있게 되었다는 것은 향후 지그비 기술개발에 많은 가능성이 추가된 것이다.

TinyOS는 재사용이 용이하도록 컴포넌트 기반으로 소프트웨어를 구현하기 때문에 다양한 하드웨어에 대한 추상화를 지원한다. 상위 응용 소프트웨어는 추상화 레이어(layer)를 이용하기 때문에 하위 하드웨어에 상관없이 다양한 플랫폼에서 동작할 수 있다. 텔로스와 MicaZ 플랫폼은 서로 다른 마이크로컨트롤러를 사용하기 때문에 무선 통신 프로토콜 하위 레이어는 서로 다르지만, 상위에서는 같은 추상화 레이어를 사용한다.
센서 네트워크 임베디드 소프트웨어 기술동향 91

<그림 13> 다양한 플랫폼에서의 IEEE 802.15.4

<그림 13>은 두 개의 플랫폼에 사용된 프로토콜 스택을 도식화했다. 두 플랫폼은 하드웨어에 의존적인 하위 부분을 제외하고 동일한 코어로 구성하여 사용한다. HPLCC2420 코어는 CC2420 칩과 마이크로컨트롤러 사이의 인터페이스가 플랫폼별로 다르기 때문에 따로 구현되어야 한다. 그림 아래 부분에 두 플랫폼에서 서로 다르게 구현된 부분을 도식화했다. 하드웨어의 구성이 서로 다른 플랫폼이지만 HPLCC2420C, TimerC 코어는 사용하는 상위 코어는 하드웨어의 변경에 부합없이 추상화되어 있는 동일한 기능을 사용할 수 있다. 소형 센서 네트워크 시스템의 하드웨어의 종류가 다양해질 것이기 때문에 TinyOS의 추상화 지원은 큰 장점으로 보이며, 모토롤라 HCS08 마이크로컨트롤러도 지
유비쿼터스1원할 예정이라 한다.

TinyOS에서는 IEEE 802.15.4 RFD(reduced function device) 구현이 진행 중이며, 텔로스와 MicaZ는 TinyOS 1.1.7 배포본부터 지원될 예정이다.

나. TRON

일본에서는 TRON(The Real-time Operating system Nucleus)을 중심으로 한 임베디드 플랫폼 표준화가 진행 중이다. 1984년부터 시작한 TRON 프로젝트를 통해 TRON VLSI 칩셋 하드웨어와 운영체제인 ITRON(Industrial TRON)에 이르기까지 실시간 임베디드 소프트웨어 규격을 제정, 산업체에서는 이를 준수하는 제품을 개발하는 형태로 진행하였다.

다. 저전력 멀티-홉(multi-hop) 네트워크 기술

무선 멀티-홉 네트워크는 현재 산업화에 가장 장애가 되는 부분이다. 많은 기술개발이 진행되어 연구단계에서는 많은 결과물이 나오고 있지만, 다양한 실제 환경에 적용할 수 있는 기술의 구현이 아직 많이 진행되지 않았다.

센서 네트워크 분야에는 다양한 애드-호(ad-hoc), 멀티-홉 라우팅(routing) 알고리즘들이 연구되고 있다. TinyOS에도 다양한 종류의 라우팅 소프트웨어들이 제안되고 있고, 특정에 따라 트리 방식, 인트라-네트워크(intra-network) 방식, 선전(dissemination) 방식, 세 가지로 구분할 수 있다. 트리방식은 센서 노드들이 데이터를 수집하여 정해진 루트(root) 노드로 데이터를 전송하는 방식이다. Habitat Monitoring\(^{12}\) 애플리케이션 구현은 트리 방식의 라우팅을 사용한다. 선전 방식의 라우팅은 대부분의 애플리케이션에 사용되는데, 패킷을 전체 네트워크에 퍼뜨리거나(broadcast), 네트워크 내의 데이터 프로세싱을 위한 캐시, 컨트롤, 재구성 패킷의 전송에 사용된다. 지금까지 구현된 멀티-홉 알고리즘을 분석해 보면 공통부분이 있는데, 주위 노드에 대한 검색, 연결 상태 계산과 데이터 패킷 생성, 전달 기능을 사용한다.

(1) 트리 라우팅

트리 방식 라우팅은 부모(parent) 노드 정보와 루트 노드로 부터의 훅(hop) 수로 경로를 결정한다. 경로 결정을 위해 루트 노드는 브로드캐스트 방식을 사용하며, 루트 주위의 노드들도 계속해서 브로드캐스트를 반복적으로 수행하여 네트워크의 최종에 위
치한 노드까지 전달되도록 한다. 이렇게 경로가 결정되면 데이터를 전송하려는 노드들은 자신의 부모 노드에게 데이터를 전송하고, 같은 방법으로 데이터를 수신한 부모 노드들은 또다시 부모 노드에게 데이터를 전송하여 루트 노드까지 데이터가 전송된다. 주의 노드를 검색하고 유지하는 방법, 데이터 전달 방법이 트리 방식 라우팅의 주요 특징을 결정한다. 지금까지 TinyOS에 구현되어 있는 트리 방식의 라우팅은 AMROUTE, BLess, Surge, mh6, MultiHopRouter가 있다.

AMROUTE는 루트 노드가 주기적인 비콘(beacon)을 전송하여 라우팅 경로를 구성한다. 단순히 현재의 부모 노드 정보만으로 경로를 구성하며, 빠른 시간 안에 회신하는 노드가 경로로 구성된다. 이와 반대로 BLess 방식은 주위 노드의 데이터 전송 경로를 참조하여 자신의 부모 노드를 결정하고 데이터를 재전송 한다. 루트 노드는 BLess 패킷을 주기적으로 전송하지만 이 패킷이 재전송 되지는 않는다.

Surge도 비콘을 사용하지 않으며 부모가 될 가능성이 있는 노드들의 정보를 이용하지만 패킷 전송 성공에 따른 링크 상태(link quality)와 흡의 수를 이용하여 경로를 결정한다. mh6와 최근 MultiHopRouter는 주변 노드들을 기준에 따라 1차 선정한 후 그 노드들이 루트까지 전송하는데 필요한 비용(cost)을 계산하여 경로를 구성한다. Surge와 MultiHopRouter는 근래에 802.11 네트워크의 8حك 라우팅에 제안된 비대칭 링크, 주변 노드 제어와 비슷한 특징을 갖는 출력 큐잉(output queuing)과 재전송 기법을 추가했다.
트리 방식의 라우팅이 많은 이유는 네트워크 구성과 데이터 전송이 알고리즘의 복잡도가 높지 않기 때문이다. 반면 장기간 안정적으로 동작하는 네트워크 구성을 위해서는 주의가 필요하다.

(2) 인트라-네트워크 라우팅

DSDV, AODV, Directed Diffusion 같은 에드-호 트방식의 라우팅 알고리즘들이 간략화되어TinyOS상에 구현되었다. 인텔(Intel)은 인터넷의 라우팅에 사용하는 방식을 채용하여 센서 네트워크에 적용하였다. 기본 알고리즘은 비슷한 방법으로 사용하였으나, 최종 결과로 구성되는 네트워크 형태는 다르게 구성된다. 주변 검색과 관리는 동일한 방법이지만 최종 네트워크 경로는 트리 방식처럼 한방향의 경로만 구성한다. TinyDiffusion은 데이터를 요구한 노드 중심으로 네트워크 경로를 구성한다. GPSR은 각 노드의 물리적 위치 정보를 이용하여 주변 노드에 오류가 발생했을 때, 문제없이 라우팅이 가능하도록 네트워크를 구성한다.

인트라-네트워크 방식은 지금까지의 TinyOS 애플리케이션들에서는 많이 고려되지 않은 방식이나, 다양한 디지털 가전기기로 구성되는 미래 홈 네트워크에 사용되기 위해서는 새로운 알고리즘이 구현되어야 한다.

(3) 브로드캐스트 라우팅

많은 애플리케이션들은 네트워크의 모든 노드들에게 데이터를 전송할 수 있는 신뢰성 있는 브로드캐스트가 필요하다. 또한 TinyOS에서는 새로운 소프트웨어를 무선 통신으로 전송하는 애플리케이션이 있는데 이런 경우에는 신뢰성 있는 멀티-홈 통신이 중요하다.

브로드캐스트의 방법으로는 단순한 재전송 방법과 필요할 때
전송하는 두 가지 방법으로 나누어진다. 단순한 재전송 방법은 루트 노드에서 데이터를 전송하면 수신한 노드가 재전송하는 방법으로 빠른 네트워크 구성에는 효과적이지만 통신 상태가 좋지 않은 곳에 위치하거나 무선 통신 총이 많은 노드는 네트워크에서 참여하기 위해 시간이 걸리는 단점이 있다. 필요시에 전송하는 방법은 노드들이 자신의 정보를 주기적으로 주위 노드와 주고받으며 실제 전송할 데이터가 발생했을 때만 전송하는 방식이다. TinyOS에 구현되어 있는 Mate 버추얼 머신(virtual machine)이 새로운 코드를 전송할 때 이런 방식을 사용한다. 이 방식은 불필요한 재전송을 막을 수 있기 때문에 에너지 효율이 좋다.

TinyDB 애플리케이션은 혼합방식을 사용했는데, 초기에는 빠른 네트워크 구성을 위해서 재전송 방식을 사용하고, 이 후에는 네트워크에 참여하지 못한 노드를 네트워크에 참여시키기 위해 방식을 전환한다.

(4) 멀티-홉 네트워크 구현 동향

MultiHopRouter, TinyDiffusion, GPSR, BVR등의 멀티-홉 네트워크 알고리즘들은 경로로 정해질 가능성이 있는 이웃 노드 검색과 관리 테이블 구성이 공통적인 동작이다. 이 정보들은 처음 라우팅 경로를 결정할 때와 주위 노드의 생성 또는 소멸 등의 네트워크 환경이 변화했을 때 경로 재설정에 사용된다. 이웃 노드 테이블에는 노드의 주소, 연결 상태, 홀 수 같은 라우팅 메타 데이터를 포함한다. 링크 상태는 라우팅 경로를 결정할 때 사용되며, 홀 수는 테이블을 관리하는데 사용된다.

TinyOS에서 초기에 구현된 멀티-홉 라우팅 프로토콜들은 전
송 오류가 심했지만\(^{20}\), 연결 상태 예측기(link state estimator)를 사용한 MultiHopRouter, BVR, TinyDiffusion, TinyOS DSDV에서는 이 문제들이 해결되었다\(^{21}\). 그리고 이들 멀티-홈 라우팅에서는 Send와 Intercept 인터페이스를 사용하기 시작했는데, 패킷 생성에 효율적인 getBuffer 커맨드와 재전송해야 하는 패킷을 수신했을 때 호출되는 Intercept 이벤트가 포함되어 있어서, 노드들은 데이터의 재전송 전에 데이터 처리가 가능해졌다. 또한 네트워크 스택 하부에 이웃 노드의 패킷을 수신할 수 있는 기능을 추가하여 더 정확한 주변 정보를 수집할 수 있다. 전송 큐, 출력 큐, 생성 큐, 전달 큐 등을 추가하여 전송 효율도 높였다.

TinyOS에 구현되어 있는 라우팅 프로토콜들은 초기와는 달리 주 변 노드에 대한 데이터를 생성, 자신의 것이 아닌 패킷의 수신, 메시지 큐의 사용으로 지연시간, 패킷 손실 등의 단점을 극복하였다.

라. 국내 기술 개발 현황

국내는 정부를 중심으로 한 기술 개발이 진행 중이다. 전자부품 연구원에서는 무선 센서네트워크 칩을 개발하고 있으며, Zigbee와 TinyOS 기술을 지원한 계획을 갖고 있다. 한국전자통신 연구소에서는 센서 네트워크 기술을 활용한 유비쿼터스 컴퓨팅 서비스 및 에이전트 기술을 개발 중이며 nano-Qplus라는 초소형 운영체제 소프트웨어를 개발하여 기술이전 하였다. 유비쿼터스 컴퓨팅 네트워크 프론티어 사업단에서는 미래형 초소형 센서네트워크 칩 및 관련 서비스 소프트웨어를 개발 중이다.
맥스포(www.maxfor.co.kr)는 전자부품연구원에서 TinyOS를 지원하는 하드웨어 플랫폼(TIP30C) 기술을 이전받아 판매 중이며, 한국전자통신연구원의 기술을 이전받은 옥타콤(www.octacomm.net/) 그리고 라디오펄스(www.radiopulse.co.kr), 오렌지로직 등의 회사들이 센서 네트워크 하드웨어를 및 솔루션을 판매 또는 계획 중이다.

### II. 기술 특성 분석

TinyOS는 센서 네트워크 전반에 대한 시스템 기술을 실제적 구현을 통해 주도하고 있다. TinyOS에 사용되는 nesC 프로그래밍 언어는 새롭게 개발하여 프로그래밍 난이도를 낮출 수 있는 기능을 제공하고 컴포넌트 기반으로 기존 모듈의 재사용성을 증가 시켰고, 저전력 무선 네트워크 구성을 요구사항에 적합한 아키텍처 및 통신 기법들을 제안한 것 등은 새로운 기술이 안정화 되는데 많은 기여를 하였다.

유비쿼터스 컴퓨팅 네트워크 환경에서는 다양한 기기와 사물들이 서로 네트워크를 구성하고 무선 통신을 하게 된다. 지금까지와는 다른 복잡한 저전력 네트워크 통신을 지원해야 하는 것이 차세대 네트워크에 대한 중요한 요구사항이다. TinyOS 프로젝트에서는 다양한 네트워크 프로토콜들이 개발되었으며, 일부는 원격 모니터링 서비스에 실제 사용되었다. 지금까지의 컴퓨팅 기술과는 다르게 TinyOS에서는 무선 네트워크를 중요한 요소로 다루고
있으며, 실제 TinyOS 내부에는 무선 통신, 네트워크 프로토콜의 최적화가 잘 이루어져 있다. 반면 운영체제 커널 스케줄러는 무선 네트워크를 지원할 수 있도록 만들어 놓아서 고급 기능들이 포함되어 있지만, 저전력 무선 네트워크라는 측면에서는 효율적으로 사용될 수 있는 장점이 있다. 미국의 많은 기관에서 TinyOS를 이용한 기술개발을 추진하고 있으며, 대부분 무선 네트워크, 서비스 기술을 개발하고 있다.

일본의 경우는 가전제품 등에 사용할 수 있는 여러 등급의 임베디드 운영체제 기술을 개발하고 있다. 운영체제 커널을 제품에 최적화할 수 있도록 규격을 제정하였고, 실제 많은 기기들에 적용되고 있다.

일본과 미국에서의 연구개발의 차이점은 미국은 네트워크 중심으로, 일본은 운영체제의 최적화를 중심으로 진행된 것으로 보인다. 미국은 새로운 무선 네트워크 시장 창출이라는 면에서는 어느 정도 가시적인 결과를 보여주고 있고, 일본은 여러 분야에 사용될 수 있는 더욱 최적화된 소프트웨어 기술 발전을 보여주고 있다.

Ⅳ. 결론 및 전망

센서 네트워크 기술은, 작고 단순한 하드웨어를 기반으로 소형 소프트웨어 개발을 지향하고 있기 때문에 고급 기술 창조를 통한 신산업의 창출이라는 기술적 기대를 실현하기에는 구현 난이도에 대한 최적의 견해가 많은 것 같다. 그러나 기술적 난이도와는 상
유비쿼터스 1

관념이 새로운 컴퓨팅 개념의 도출과 실제 구현을 지향하는 기술 개발 방향은 매우 긍정적으로 평가받을 수 있는 추진 방향이다.

센서 네트워크 같은 저전력 무선 네트워크가 실제 산업과 생활에 적용되고 상용화되기 위해서는 안정되고 상호운용이 가능한 네트워크 프로토콜의 실제 구현이 요구되는데, 이런 요구사항을 해결할 수 있는 방법은 지그비 규격의 제품이 개발되거나 TinyOS를 기반으로 한 기술구현 등으로 예상된다. IEEE 802.15.4 기반의 하드웨어를 이용한 무선 통신 프로토콜이 TinyOS상에서 구현되었기 때문에 하위 기술에 대한 호환성이 보장되었으며, 지금부터는 네트워크 부분의 기술 구현이 본격적으로 진행될 것으로 보인다.

TinyOS도 지그비 얼라이언스(Zigbee Alliance)와 비슷한 성격의 TinyOS 얼라이언스를 구성할 계획이다. 많은 동일 기술요소를 포함하는 두 진영의 경쟁과 상호 보완은 새로운 기술의 산업화를 가속화 할 것으로 기대된다. 단기적으로는 무선 네트워크 시장의 확대를 가져올 것이며, 장기적으로는 유비쿼터스 컴퓨팅에서의 끊김 없는 네트워크를 구성할 수 있는 기반 기술로 사용될 것이다.
참고자료

5. J. Elson, S. Bien, N. Busek, V. Bychkovskiy, A. Cerpa,