무선 센서 노드를 위한 FSM 기반 운영체제의 구현†

(Implementing Finite State Machine Based Operating System for Wireless Sensor Nodes)

하 승 현*, 김 태 형**
(Seung Hyun Ha and Tae-Hyung Kim)

요 약 무선 센서 네트워크는 단거리 무선 통신으로 연결된 지능 센서가 사람과 실세계 객체간의 효과적인 매개자 역할을 하므로 최근 유비쿼터스 컴퓨팅 환경을 가능하게 할 수 있는 핵심적인 기술 중 하나로 각광을 받고 있다. 무선 센서 네트워크는 대량으로 분산된 극도의 내장형 시스템으로 볼 수 있다. 이 시스템은 분산 시스템으로서 병행성과 비동기적 이벤트 처리 능력에 대한 요구사항과 함께 내장형 시스템으로서 자원제한성에 따른 심각한 요구사항을 갖고 있다. 이러한 일견 상충하는 두가지 요구사항을 갖는 무선 센서 네트워크의 운영 환경과 구조는 시스템 개발자에게 매우 독특한 어려움을 제기하고 있으므로 우리는 유한상태기계에 기반을 둔, 매우 새로운 형태의 센서 네트워크용 운영체제를 제안한다. 본 논문에서는 센서 네트워크의 특성을 감안한 설계 목표를 명시하고, 간결하고 효율적인 상태기반 운영체제인 SenOS 설계와 구현의 핵심 사항을 제시한다. 또한 이러한 SenOS가 심각한 자원제한성을 갖는 센서 노드 상에서 원하는 반응성을 갖으면서도 적은 비용으로 동적 재구성이 가능하다는 것을 설명한다. 이러한 성능은 대표적인 센서 노드용 운영체제인 TinyOS의 성능평가에 사용된 벤치마크 프로그램을 수행하고 그 결과를 TinyOS의 경우와 비교한다.

핵심주제어 : 무선 센서 네트워크, 운영체제, 유한상태기계, 성능평가

Abstract Wireless sensor networks have emerged as one of the key enabling technologies for ubiquitous computing since wireless intelligent sensor nodes connected by short range communication media serve as a smart intermediary between physical objects and people in ubiquitous computing environment. We recognize the wireless sensor network as a massively distributed and deeply embedded system. Such systems require concurrent and asynchronous event handling as a distributed system and resource-consciousness as an embedded system. Since the operating environment and architecture of wireless sensor networks, with the seemingly conflicting requirements, poses unique design challenges and constraints to developers, we propose a very new operating system for sensor nodes based on finite state machine. In this paper, we clarify the design goals reflected from the characteristics of sensor networks, and then present the heart of the design and implementation of a compact and efficient state-driven operating system, SenOS. We

† 본 연구는 경기도의 경기도지역협력연구센터(GRRC) 사업의 일환으로 수행하였음. [GRRC한양2010-B04]
* 삼성전자 무선사업부, 제1저자
** 한양대학교 컴퓨터공학과, 교신저자

describe how SenOS can operate in an extremely resource constrained sensor node while providing the required reactivity and dynamic reconfigurability with low update cost. We also compare our experimental results after executing some benchmark programs on SenOS with those on TinyOS.

Key Words : Wireless Sensor Network, Operating System, Finite State Machine, Performance Evaluation

1. 서 론

무선 센서 네트워크는 각종 센서에서 수집한 빛, 소리, 온도, 움직임과 같은 물리적인 환경 데이터를 무선으로 수집할 수 있도록 구성한 네트워크를 말하며 저전력 마이크로컨트롤러와 무선 통신 모듈, 통합된 시스템 장치들을 내장한 센서 노드를 이용하여 구성하게 된다[1]. 실제 연구에 사용되는 센서 노드를 살펴보면 8비트 마이크로컨트롤러, 5-50m의 통신 범위를 갖는 무선 모듈, 128KB 이내의 코드 메모리, 20KB 보다 적은 데이터 메모리가 장치되어 있고 버클리 대학에서 개발한 WeC, rene2, mica, micaz 등과 같은 장비들이 대표적인 센서 노드이다[2,7]. 이러한 센서 노드는 기존의 컴퓨팅 시스템과는 달리 제한된 시스템 자원과 열악한 환경 속에서 동작하며 수많은 센서 노드로 무선 센서 네트워크를 형성하여 산업, 군사, 가정, 환경 감시 등의 다양한 응용에 적용할 수 있다[22, 23].

이와 같이 범용적인 서비스를 제공하기 위해선 다양한 응용 프로그램을 개발하고 실행할 수 있는 환경, 수집된 정보를 목적지로 전달하기 위한 노드 간 통신, 제한된 시스템 자원을 효율적으로 관리할 수 있는 기능들이 필요하다. 또한 무선 센서 네트워크의 특성상 응용 프로그램이 설치된 센서 노드를 센서 필드에 배치하고 나면 센서 노드는 쉽게 수거되기 어렵고 재프로그래밍을 위한 물리적인 연결이 힘들다. 따라서 응용의 변화에 따른 업데이트가 쉽지 않게 되는데 이를 해결하기 위해서는 동적 재구성 기능의 지원이 필수적으로 요구된다[3,7].

이렇게 범용적인 시스템의 요구 조건뿐 아니라 내장형 시스템의 특성 또한 고려되어야 하는 환경에서 센서 노드용 운영체제에 대한 연구가 이루어졌다. 현재 개발되고 연구 중인 센서 운영체제들은 센서 노드의 특수성에 기반하여 별도의 설계를 하기보다는 기존 운영체제의 설계 개념과 운영방식을 센서 네트워크 환경에 적응(adapt)시키려는 노력이 일반적이다. 버클리 대학에서 개발한 TinyOS[2]를 대표적인 예로 들 수 있으며, 그 외 SOS[3], MANTIS[4], PEEROS[5] 등과 같은 센서 운영체제에 대한 연구가 발표되었다. 그러나 이와 같은 기존의 센서 운영체제들은 (1) 센서 노드에 최적화시켰지만 동적 재구성 기능을 지원하지 못하거나 (예: [2]의 경우), (2) 응용 프로그램의 실행 구조상 동기화를 위해 복잡한 메커니즘을 사용하기도 하고 (예: [4,5]), (3) 가상머신 기법을 적용하여 동적 재구성 기능을 지원하지만 상대적으로 시스템 자원의 사용 비용이 높아지는 문제를 갖고 있기도 한다(예: [7]).

실행환경이 전혀 다른 일반적인 형태의 운영체제 개념을 축소하는 방향으로 진행된 위와 같은 기존 연구와는 달리, 센서 노드의 응용에 적합한 유한상태기계(Finite State Machine) 기반의 새로운 운영체제에 대한 개념 설계가 발표된 바 있다[6]. 본 논문에서는 새로운 패러다임으로 설계된 SenOS를 128KB의 ROM과 2KB의 RAM과 함께 8비트 마이크로컨트롤러를 장착한 ATmega128L 노드에 FSM 기반의 실행모델을 적용하고, 응용의 변화에 대처할 수 있게 동적 재구성 기능을 제공하는 SenOS[6]를 구현한 내용과 함께, 코드크기, 전력소모, 반응성과 같은 다양한 요구조건들을 어떻게 만족시키는지를 대표적인 센서용 운영체제인 TinyOS와 대조한 결과를 발표한다. 본 논문의 2절에서는 기존에 개발된 센서 운영체제의 특징과 문제점들을 살펴보고 3절에서는 구현된 FSM 기반의 운영체제인 SenOS의 특징과 구조를 설명한다. 4절에서는 SenOS의 성능 평가를 위해 실험한 내용과 결과를 분석하고 5절에서 결론을 맺는다.

2. 관련연구

2.1 센서 운영체제

센서 네트워크 노드를 위한 운영체제는 이벤트 구동 방식을 사용하는 운영체제[2,3,6]와 멀티테스킹[5]과 멀티스레드[4]를 사용하는 운영체제로 나누어 볼 수 있다. 이 두 방식의 두드러진 차이점은 스케줄링 방식에 있다. 이벤트 구동 방식은 high-low, task-event와 같이 2-레벨의 우선순위를 갖는 FIFO 스케줄링 기법을 사용하고, 멀티태스킹과 멀티스레드 방식을 사용하는 운영체제에서는 프로세서의 타임 슬라이스를 이용하기 위해 라운드 로빈 방식의 스케줄링이나 EDFI[20]와 같은 스케줄링을 사용한다.

대표적으로 버클리에서 개발한 TinyOS[2]는 컴포넌트를 기반으로 각 모듈간 Command 및 Event를 통해서 데이터의 전송이 이루어지는 "이벤트 구동" 방식을 사용 한다. 스택 기반에 스레드 모델이 이용할 수 있는 것보다 더 많은 메모리 크기를 요구할 수도 있고 메모리 사용에 있어 정교한 메모리 관리가 필요하기 때문에 적은 자원을 이용하여 높은 병행성을 보장할 수 있기 때문이다. TinyOS를 위한 응용 개발시 프로그래밍 언어는 C를 확장한 nesC를 사용한다. nesC는 이벤트 구동 시스템을 지원하기 위한 컴포넌트 모델을 정의하고 있을 뿐 아니라, TinyOS를 재구현하는 데에도 사용되었다[21]. 컴포넌트 모델을 이용하여 소프트웨어적으로 하드웨어를 추상화하였으므로 계층별로 재사용 가능하게 모듈화된 컴포넌트를 이용하여 센서 노드용 응용 프로그램을 작성할 수 있다. 이렇게 작성된 응용 프로그램은 nesC 컴파일러를 통해 노드에 최적화된 코드를 생성하게 된다. 하지만 nesC 모듈들의 병행성(Concurrency)을 향상시키기 위해 함수 호출이 블록 되지 않는 스플릿 페이즈(split-phase)로 동작하는 구조상, 만약 태스크의 post가 실패하면 콤포넌트는 완료 이벤트를 기다리면서 영원히 블록될 수 있다는 문제점과 응용 프로그램에 사용할 컴포넌트 설계시 수행해야할 태스크를 스플릿 페이즈 구조를 고려하여 설계해야 하므로 nesC를 이용한 응용 프로그램의 개발은 매우 어려운 것으로 알려져 있다[21].

프로그래밍 모델 상의 어려움을 극복하기 위한 콜로라도 대학의 MANTIS OS(MOS)는 멀티스레드를 지원하고 응용개발 프로세스의 단순화를 통하여 센서 네트워크를 효율적으로 구축하고자 한 센서 노드용 운영체제이다[4]. MOS 커널의 설계는 전통적인 유닉스 스케줄러와 유사하며 라운드 로빈을 사용한 우선순위 기반의 스레드 스케줄링을 수행하고 멀티 스레드의 동기화 문제를 해결하기 위해 뮤텍스와 카운팅 세마포어를 지원하고 있다. C로 작성된 MOS의 표준 프로그래밍 언어와 표준 스레딩 모델은 센서 네트워크 응용 프로그램 개발과정 측면에서 일반 프로그래밍 개발 기술을 이용할 수 있기 때문에 접근이 용이하다[4]. 또한 멀티스레딩 모델은 현재 CPU를 점유하고 있는 태스크의 실행 시간이 길어지면 다음 태스크의 실행을 지연시키는 문제가 발생할 수 있는 이벤트 구동 방식의 단점을 보완할 수 있다. 하지만, 시스템 자원이 제한적인 센서 네트워크 특성상 스케줄링과 스레드를 실행하고 관리하는 운용 비용이 이벤트 구동 방식의 운영체제보다 상대적으로 높다고 할 수 있다.

SenOS는 전력소모 감소가 중요한 센서 노드의 특성상, 실행 비용이 적게 드는 이벤트 구동 방식으로 동작하도록 설계되었으며, FSM 기반의 실행 모델을 채택하였고, 동적 재구성 기능의 지원, 이벤트 구동 방식의 태스크 블록킹 최소화, 응용 개발이 쉬운 구조를 지향하여 설계하였다[6]. 결과적으로 센서 응용 프로그램의 개발과정은 센서 노드의 태스크를 상태 모델링을 통해 상태 전이 테이블로 구성하게 되고 이러한 구조는 개발자가 UML 상태차트를 기반으로한 다양한 모델링 도구를 이용하여 SenOS에서 제공하는 콜백 함수를 호출하도록 하는 방법으로 프로그램할 수 있으므로, 응용 프로그램의 개발이 매우 용이하다[6].

2.2 동적 재구성의 필요성

무선 센서 네트워크 환경에서 동적 재구성은 센서 노드가 필드에 배치되고 초기화된 이후 각각의 노드들에 설치된 응용 프로그램이 동작 중에도 수정할 수 있는 기능을 말한다. 운영체제와 응용 프로그램으로 구성된 시스템 이미지는 소스 코드 작성, 컴파일, 링크 과정을 거쳐 생성되고 유선통신을 통해 센서 노드에 프로그램 된다. 이렇게 프로그램된 센서 노드가 필드

에 배치되고 나면 동적 재구성 기능이 없을 경우 환경 변화와 목적에 따라 변할 수 있는 응용 프로그램의 업데이트가 쉽지 않게 된다[3,4,10]. 만일 동적 재구성 기능이 제공된다면 노드들이 필드에 배치된 이후에도 무선 통신을 이용하여 새로운 응용 프로그램을 추가하거나 시스템에 설치된 응용 프로그램의 변경, 오랫동안 사용되지 않는 소프트웨어 모듈의 제거를 통해 시스템 이미지 설치 이전에 예측할 수 없었던 환경 적응형 센서 네트워크 응용 프로그램의 실행이 가능하다. 하지만, 제한적인 시스템 자원을 가진 센서 노드의 특성상 이러한 재구성 기능은 업데이트에 따른 CPU 사용 비용, 무선 통신의 데이터 전송 비용, ROM과 RAM을 사용한 비용 등이 고려되어야 하며 전력 공급이 제한적인 센서 노드의 생명 주기에도 영향을 주게 되므로 재구성 비용의 최소화가 필요하다[1].

2.3 동적 재구성 방법

센서 노드의 응용 프로그램의 변화에 대처하기 위해 코드나 기능을 추가하는 동적 재구성과 관련하여 많은 메커니즘이 연구되었는데 여기에는 전체 시스템 이미지 교체[8], 가상머신의 사용[7], 로드 가능한 원시 코드 모듈의 지원[3,9] 등과 같은 방법들이 있다. TinyOS는 센서 노드에 최적화시킨 아주 작은 운영체제이지만 프로그램된 시스템이 컴파일 시점에 정적으로 링크된 바이너리 이미지이기 때문이 동적 재구성 기능을 지원하지는 않는다[3]. 그러나 센서 노드의 필드 배치 완료후, 프로그램의 변경 필요성이 대두되면, XNP[8]라는 in-network programming 방식을 이용하여 노드에 설치되어 있는 시스템 이미지 전체를 무선 통신을 통해 전송하고 교체하는 방법으로 동작중인 TinyOS 프로그램을 업데이트하는 기능을 제공한다. 업데이트 과정이 단순하지만 업데이트에는 컴파일 시간에 정적으로 링크되어 생성된 새로운 TinyOS의 시스템 이미지가 사용되고 전체 시스템 이미지를 무선 네트워크를 이용해서 전송해야 하므로 코드 업데이트 비용이 상당히 높아진다는 단점이 있다[3,10].

프로그램 코드의 전송비용을 줄이기 위해 가상머신 사용방법이 연구되었으며 TinyOS의 Mate가 여기에 해당한다. Mate는 가상머신에서 제공하는 명령어 셋트를 이용하여 응용 프로그램을 구성하며 프로그램 업데이트시 캡슐화 된 프로그램을 전송함으로써 코드 업데이트 비용과 통신비용을 낮춘다. 하지만 캡슐화된 프로그램을 해석하기 위한 새로운 연산 오버헤드를 야기시킨다. 또한 노드마다 가상머신이 설치되어야 하며 가상머신을 위한 시스템 자원이 추가로 요구된다는 단점이 있다[3].

이 밖에 SOS[3]나 Contiki[9]에서는 "로드 가능한 모듈(Loadable Modules)"들을 이용하여 동적 재구성 기능을 지원하고 있다. TinyOS의 정적 링크는 전체 시스템을 완전한 구조의 바이너리로 변환하고 컴파일시 코드가 최적화된 후 주소가 결정되므로 변경이 어렵게 된다. 따라서 시스템 교체시 전체 시스템 이미지를 다운로드 해야 하지만, SOS나 Contiki의 경우 동적 링크를 사용할 수 있도록 하여 응용의 변화에 loadable modules를 기반으로 대처할 수 있게 하였다. SOS는 모듈 로딩시 재배치 과정을 피하기 위해 컴파일 시간에 모듈을 위치 독립적인 코드(Position Independent Code)로 생성하여 업데이트에 이용한다. Contiki는 SOS와 개념적으로 동일하나 리눅스 표준파일포맷인 ELF(Executable and Linkable Format)을 이용하여 모듈을 구성하고 동작 중인 시스템 내에 ELF 로더를 두어 업데이트를 수행한다. 로드 가능한 모듈로 응용 프로그램을 구성하는 시스템 구조상 시스템이 동작 중인 상태에서도 바이너리 파일을 로드할 수 있고 전체 시스템 이미지를 교체하지 않고도 응용 프로그램에 해당하는 모듈을 업데이트할 수 있다. 이전에 센서 네트워크에서 동적 링킹은 실행시간에 대한 오버헤드, 에너지 소비, 메모리 요구로 인해 고려되지 않았지만 앞서 언급한 코드 업데이트 비용 문제들을 SOS나 Contiki의 경우 로드 가능한 모듈을 이용하여 해결하고자 하였다. 이러한 동적 재구성 기능은 Mate의 가상머신 보다 연산 오버헤드와 점유하는 메모리의 용량이 작다는 장점이 있지만 코드 업데이트 비용은 여전히 높다[3]. 이러한 자원 제한적인 센서 노드에서 동적 재구성시 소모되는 코드 업데이트 비용과 연산 오버헤드, 시스템 자원의 사용 비용을 낮추고자 SenOS에서는 교체 가능한 상태 전이 테이블 단계업데이트, 콜백 함수 단계 업데이트로 구성되는 2-레벨

동적 재구성 기능을 지원하도록 설계하였다.

3. SenOS의 특징 및 구현

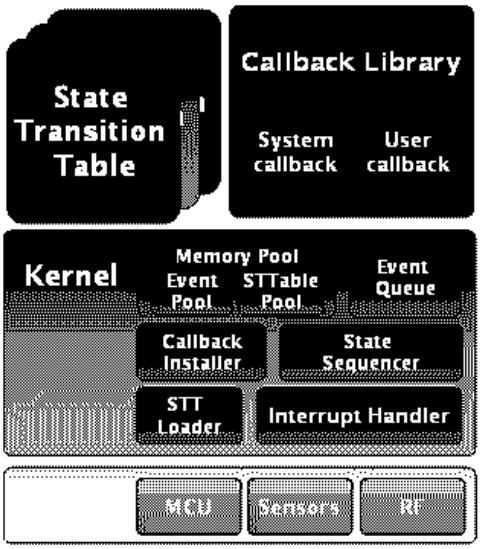
센서 운영체제로 제안된 SenOS는 FSM 기반의 실행 모델을 이용하여 설계되었고 이벤트 구동 방식으로 동작한다[6]. FSM은 발생하는 이벤트에 즉각 반응해야 하는 이벤트 구동형 시스템을 설계하고 구현하는 데 있어 효과적인 접근 방법으로 SenOS는 센서 노드의 상태가 유한한 점을 이용하였다[6,14]. SenOS는 센서 노드의 상태와 이벤트를 상태 전이 테이블로 표현하고 입력 이벤트에 따라 상태 전이와 출력을 발생시킨다. 상태 전이는 실행모델 개념상 즉각적으로(instantaneously) 일어나며 전이된 상태와 연관된 출력 함수가 호출된다. 이와 같은 실행 메커니즘을 이용하여 SenOS는 연속된 일련의 동작들을 수행하거나 FSM의 상태에 따라 입력 이벤트를 다르게 처리한다. 이러한 FSM 기반 컴퓨팅 모델은 프로그램 디자인이 쉽고, 반응이 빠르며, 프로그램이 선점되는 과정에서 저장되고 복구되는 정보가 명확하기 때문에 시스템은 효율적으로 프로그램을 정지시키고 복구시킬 수 있다는 장점이 있다.

센서 노드가 수행해야 할 작업들이 다른 노드로부터 메시지를 받거나 특정 현상에 대한 감지 또는 특정한 시간이 되어 정해진 일을 수행하는 등 이벤트의 발생에 의해 수행되는 특징을 가지고 있으며 SenOS는 이러한 이벤트 처리에 RTC(Run-To-Completion)라고도 불리는 비선점형 모델을 사용하고 있다. 비선점형 모델에서는 실행 중인 태스크가 다른 태스크에 의해 선점되지 않으며 이벤트 발생시 수행되는 태스크들이 무한 루프 형태의 스레드 프로세스와 달리 짧은 시간 동안 목적한 일을 처리하고 종료된다. 따라서 선점하는 태스크의 수행 시간을 작게 할수록 시스템의 반응성을 높일 수 있다. 또한, 비선점형 모델은 멀티스레드와 같은 선점형 모델에 비하여 컴퓨팅 파워를 절약할 수 있다. 멀티스레드 모델에서는 스레드 생성시 스택 메모리의 할당과 해제와 같은 관리가 필요하고 우선순위에 따라 의도하지 않는 변수의 수정과 같은 병행성 문제가 발생한다. 따라서 공유 변수에 대해 상호배제와 같은 동기화 메커니즘이 필요하게 되어 시스템이 복잡해지고 실행 비용이 상승하게 된다[14].

FSM 기반의 실행 모델과 이벤트 구동 방식을 사용하여 태스크를 처리하는 SenOS는 이벤트 큐로부터 입력을 받고 이를 처리하는 상태 정렬기(state sequencer), 이벤트를 저장하는 이벤트 큐, 유효한 상태 전이와 연관된 콜백 함수들을 정의하고 있는 상태 전이 테이블(state transition table), 출력 함수들을 담고 있는 콜백 라이브러리, 인터럽트를 감지하여 처리하는 인터럽트 필터, 동적으로 테이블을 재구성할 수 있는 동적 라이브러리 로더로 이루어져 있다. 본 연구에서 사용한 센서 노드는 8-bit MCU ATmega128L[17]과 실내에서 50m에 무선 송신 범위를 갖는 CC2420 (Zigbee) RF 트랜시버[18]가 장치되어 있고 내부적으로 128KByte의 프로그램 메모리, 2KByte SRAM, 4KByte EEPROM을 갖는다. 추가로 센서 데이터를 장시간 저장하는데 사용할 수 있는 512KByte의 Flash 메모리가 외부에 부착되어 있으며 온도, 습도, 조도, 적외선을 측정할 수 있는 센서가 있다.

3.1 SenOS의 내부 구조

이벤트를 처리하는 상태 정렬기와 이벤트 큐로 이루어진 커널, 상태 전이 테이블, 출력 함수들을 담고 있는 콜백 라이브러리는 기존에 발표된 SenOS와 같으며, 그림 1은 본 연구를 통해 개선된 SenOS의 전체 구조를 나타낸다. 위 개념을 효과적으로 구현하기 위해 그림에서와 같이 커널은 콜백 함수를 시스템에 등록하는 콜백 인스톨러, 이벤트 생성 시 메모리를 할당해 주는 이벤트 풀, 동적 재구성시 새로운 상태 테이블을 저장하는데 필요한 메모리를 할당해 주는 테이블 풀, 동적으로 테이블을 관리하기 위해 사용되는 테이블 로더 등이 추가로 구현 되었다. 하드웨어 계층에서 발생되는 이벤트는 핸들러를 통해 큐에 등록되며 상태 정렬기에서 상태 전이 테이블을 검사하여 콜백 함수를 호출하게 된다. 그리고 모듈화 과정을 통해 응용에 따라 선택적으로 SenOS를 구성할 수 있도록 하였다.



<그림 1> SenOS의 구조와 컴포넌트들

3.2 SenOS의 동적 재구성

위와 같은 H/W-level 및 function-level 정적 재구성 외에도 SenOS의 응용 프로그램에 해당하는 상태 전이 테이블을 교체하거나 추가하는 기능을 제공함으로써 동적으로 응용프로그램을 재구성하는 것도 가능하다. SenOS에서의 센서 응용 개발은 태스크를 상태 모델링하여 상태 전이 테이블을 생성하고 필요한 콜백 함수를 선택하는 과정으로 진행된다.

3.3 SenOS의 동적 재구성

설치시 H/W-level 및 function-level에 따른 정적 재구성 외에도 SenOS의 응용 프로그램에 해당하는 상태 전이 테이블을 교체하거나 추가하는 기능을 제공함으로써 동적으로 응용 프로그램을 재구성하는 것도 가능하다. SenOS에서의 센서 응용 개발은 태스크를 상태 모델링하여 상태 전이 테이블을 생성하고 필요한 콜백 함수를 선택하는 과정으로 진행된다. 이때 생성되는 상태 전이 테이블과 콜백 함수는 센서 노드의 기능을 나타내며 응용의 변화에 따라 시스템에 이를 반영할 수 있는 동적 재구성 기능과도 밀접한 관련이 있다. SenOS은 상태 전이 테이블 변경과 콜백 함수의 교체를 이용한 2-단계의 동적 재구성을 지원하고 있다. 업데이트 요청시 센서 노드에 새로운 상태 전이 테이블이 참조할 콜백 함수가 없는 경우 새로운 콜백 함수를 노드로 내려보내야 하기 때문에 업데이트 비용 절감에 대한 기대치는 낮아지겠지만 필요한 기능을 추가할 수 있고 콜백 함수가 시스템에 있는 경우 상태 전이 테이블 교체만으로도 업데이트가 가능하게 되어 업데이트 비용을 줄일 수 있게 된다.

	Current State	Input event	Destination State	Action
APP_1	IDLE	START_SENSING	SENSE	select sensor, get sensed data
	SENSE	SENSING_DONE	IDLE	make packet for send
	IDLE	SEND_SENSED_DATA	SEND	send data packet
	SEND	SEND_DONE	IDLE	do nothing
	IDLE	START_TIMER	TIMER	start timer
	TIMER	DO_NOTHING	IDLE	do nothing
	IDLE	SET_TIMER	TIMER	set timer period
	TIMER	SET_TIMER_DONE	IDLE	do nothing
APP_2	IDLE	EFLSH_READ_DATA	STORE	read sensed data to external flash
	STORE	EFLSH_READ_DONE	IDLE	make packet for send
	IDLE	EFLSH_WRITE_DATA	STORE	write sensed data to external flash
	STORE	EFLSH_WRITE_DONE	IDLE	do nothing
	IDLE	RECV_PACET	CMD	check recv packet and create event
	CMD	DO_NOTHING	IDLE	do nothing

<그림 2> 상태 전이 테이블의 변경을 통한 동적 재구성

3.3.1 상태 전이 테이블과 동적 재구성

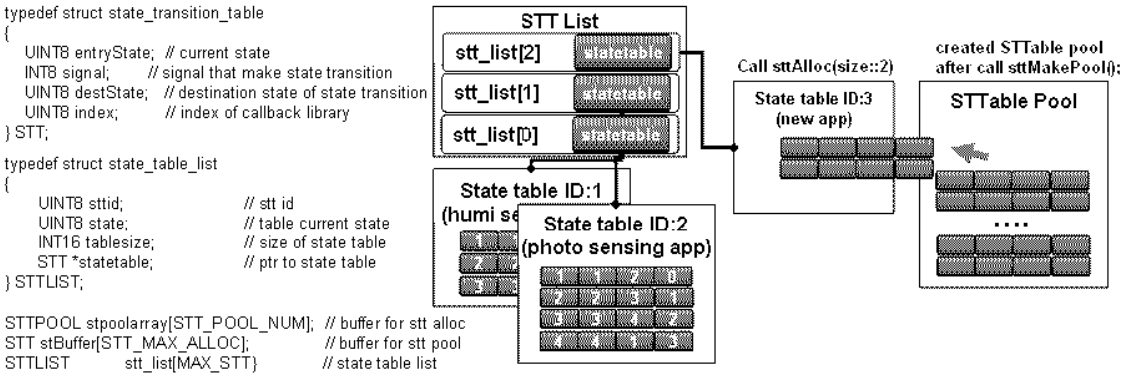
이벤트 SenOS에서 상태 전이 테이블은 센서노드가 수행하는 응용 프로그램에 해당하며 센서 노드가 수행할 동작들이 기술되어 있다. 상태 전이 테이블을 작

성하기 위해서는 센서 노드의 동작을 표현하는 상태 모델링이 필요하고 상태 모델링을 통해 응용에 필요한 이벤트와 상태를 파악하여 상태 전이 테이블로 작성한 후 시스템에 로딩하면 이벤트 발생시 테이블에 등록된 태스크가 수행된다. 상태 전이 테이블은 현재 상태, 이벤트, 다음 상태, 액션 필드로 이루어져 있고 그림 2는 조도를 측정하여 저장하고 측정된 데이터를 무선을 통해 전송하는 상태 전이 테이블이다.

그림 2에서와 같이 타이머 이벤트를 이용하여 데이터를 수집하고 전송하는 응용 프로그램(APP_1)이 설치되어 있던 센서 노드들이 싱크 노드에서 전달된 명령으로 수집한 데이터를 저장하고 저장된 데이터를

call senosChangeSTT(void);

그 후 수신 데이터에 등록된 새로운 테이블의 크기만큼 메모리를 할당하고 상태 전이 테이블 데이터를 추출하여 각 항목을 설정한 후 상태 전이 테이블 로더가 시스템에 새로운 테이블을 등록 시키고 재시작하면 업데이트가 완료된다. 상태 전이 테이블은 그림 3과 같은 구조이며 업데이트시 사용되는 테이블풀 공간은 동적 메모리 사용에 따른 문제점을 피하기 위해 개발자가 지정한 연속된 메모리 공간(정적 메모리 공간)을 확보해 두고 메모리 요청시 할당해 준다.



<그림 3> 상태 전이 테이블 구조와 관리

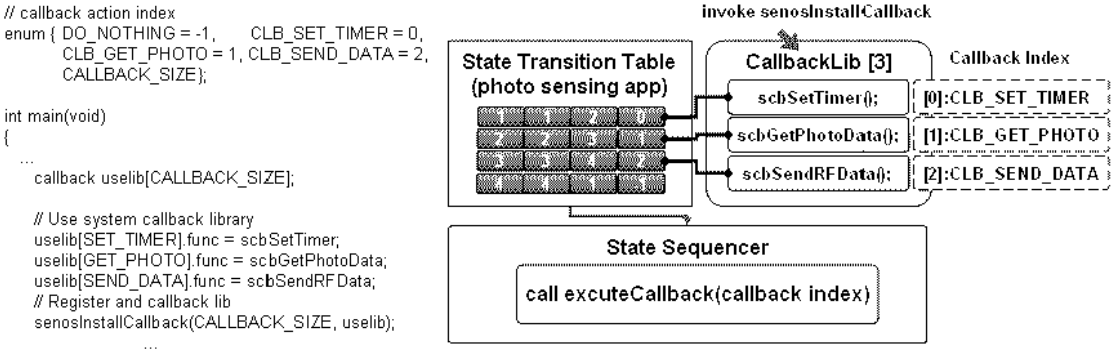
전송하는 새로운 응용 프로그램(APP_2)으로 변경되기 위해선 상태 전이 테이블이 동적 재구성시 업데이트 될 수 있게 교체 가능해야 하며 적절히 관리할 수 있는 모듈이 필요하다. 이를 위해 SenOS는 동적 재구성 기능을 사용할 때, 새로운 상태 전이 테이블 등록에 필요한 메모리를 할당해 주기 위한 테이블 풀을 만들고 메모리 할당과 해제, 테이블 교체를 수행하는 함수들을 활성화 시킨다. 상태 전이 테이블 업데이트는 시스템이 응용 업데이트 메시지를 수신할 경우 상태 전이 테이블 교체 이벤트의 발생으로 시작하게 된다. 상태 정렬기에서는 이벤트 큐에 등록되어 처리할 이벤트가 STT_CHANGE_EVNT이면 시스템을 안전한 상태로 변경하여 잘못된 연산을 방지하고 상태 전이 테이블 교체 실행을 위한 함수를 다음과 같은 방법으로 호출한다.

3.3.2 콜백 함수와 동적 재구성

센서 노드는 정보 수집, 데이터 전송, 데이터 저장과 같은 여러 가지 태스크들을 수행한다. SenOS에서는 이렇게 센서 노드가 수행해야 할 여러 태스크를 분할하여 독립된 하나의 태스크로 나타내고 콜백 함수로 처리한다. 콜백 함수가 독립된 태스크로 구성되어 있으면 이벤트 발생시 상태 전이 테이블과 연계된 콜백 함수의 호출만으로 해당 태스크를 처리할 수 있게 되고 상태 전이 테이블을 이용한 응용 개발에 사용하기 편리하다는 장점이 있다. 콜백 함수는 조도나 온도 측정, 무선 데이터 전송, 배터리 체크를 수행하는 드라이버 함수와 라우팅 경로 계산, 측정된 센서 값의 변환, C 표준 라이브러리와 같은 함수들을 이용하여 구성하고 형태는 아래와 같다.

void callbackLibraryProtoType(INT8 event,
*void * param);*

SenOS는 드라이버와 밀접한 관련이 있는 콜백 함수부터 라우팅 경로를 계산하는 콜백 함수에 이르기까지 다양한 콜백 함수들을 지원하고 있으며 SenOS에서 기본적으로 제공하는 콜백 함수, 즉 시스템 콜백 함수 이외에도 개발자가 필요한 기능을 추가하여 작성한 콜백 함수(유저 콜백 함수)를 센서 응용 프로그램 개발시 사용할 수 있다. 그림 4에서와 같이 상태 전이 테이블에는 상태 전이시 호출되는 콜백 함수의 정보가 등록되어 있다. 상태 전이 테이블에 등록되는 콜백 함수의 정보는 콜백 함수에 접근할 때 사용하는 인덱스 값이며 시스템 초기화 과정에서 결정된다. 인덱스 값은 상태 시퀀서에서 상태 전이시 호출할 콜백 함수에 접근할 때 사용하며 실행할 콜백 함수를 추가적인 검색이나 특별한 알고리즘 없이 빠르게 접근할 수 있도록 해준다. 그리고 상태 전이 테이블에서 참조할 콜백 함수의 등록은 senosInstallCallback() 함수를 통해 이루어지며 콜백 함수들의 주소는 리스트를 통해 관리한다. 동적 재구성시 새로운 상태 전이 테이블에서 사용할 콜백 함수가 센서 노드에 없는 경우 상태 전이 테이블 업데이트만으로 응용의 변화에 대처하기 어렵게 되고 이 경우 새로운 콜백 함수를 노드로 내려보내야 한다. SenOS의 콜백 함수 업데이트는 동적 링크를 이용하고 새로운 콜백 함수를 컴파일 후 동적으로 재배치할 수 있는 형식으로 만들고 코드, 데이터, 심볼 테이블을 추출한다. 그 후 싱크노드에서 계산된 재배치 위치, 함수와 전역 변수의 참조 주소 정보를 간단한 형태의 헤더로 만든 후 패킷에 포함하여 코드 데이터와 함께 전송한다. 이를 수신한 노드에서는 패킷의 헤더 정보를 보고 코드와 데이터를 링크하고 재배치를 수행하여 새로운 콜백 함수를 업데이트하게 된다[9,10].



<그림 4> 콜백 함수의 등록과 동작

3.4 이벤트 관리

이벤트는 FSM의 상태를 결정하는 주체로써 시스템 내에서 가장 빈번하게 처리된다. 이러한 이벤트들은 시스템이 구동 중일 때 얼마나 발생할지 예측하기 어려우므로 이벤트 생성 요청 시에 동적으로 메모리를 할당할 필요가 있다. 일반적으로 센서 노드들은 시스템 리셋 없이 오랫동안 안정적으로 실행되어야 하는데 RAM을 사용한 힙 기반 메모리 할당은 단편화, 메모리 누수, 복잡한 힙 검사 알고리즘으로 인한 실행 시간의 오버헤드, 댕글링 포인터와 같은 문제점을 야기하여 시스템의 안정성을 낮추게 된다[13]. 이를 해결하기 위해 시스템을 초기화할 때, 개발자가 시스템 운영에 필요한 메모리를 설정하도록 하고 정적 메모리 공간을 사용하였다. 이렇게 지정한 연속된 메모리 공간을 이벤트의 크기만큼 나누어 미리 메모리 풀(Memory pool)을 만들어 놓고 이벤트 생성시 이를 할당 해주며 사용이 완료된 이벤트는 사용한 메모리 블록을 반환하게 된다. 일반적으로 SenOS에서 메모리

풀을 생성하여 사용하는 경우는 많지 않지만 응용 프로그램 과정에서 이벤트 발생시 수행되는 태스크에 필요한 인자를 처리할 때 사용할 수 있다.

이벤트 구동 방식의 SenOS에는 타이머 이벤트와 같이 동기적으로 발생하거나 무선패킷의 수신처럼 비동기적으로 발생하는 이벤트를 저장할 이벤트 큐가 필요하게 되며 커널에 응용 프로그램의 이벤트 사용과 처리 형태 따라 사용할 수 있는 큐가 포함되어 있다. 이벤트 큐는 이전 시스템과 달리 2-레벨의 우선순위를 갖는 FIFO 방식으로 개선하였다. 일반적인 이벤트들은 FIFO 방식으로 저장하며 긴급하게 처리되어야 할 이벤트는 트는 이벤트 생성시 LIFO 방식으로 큐에 삽입할 수 있도록 함으로써 긴급한 처리를 위한 프로그래밍이 가능하도록 하였고 eventCreateLIFO(sig) 함수를 이용하여 실행할 수 있다.

4. 평가

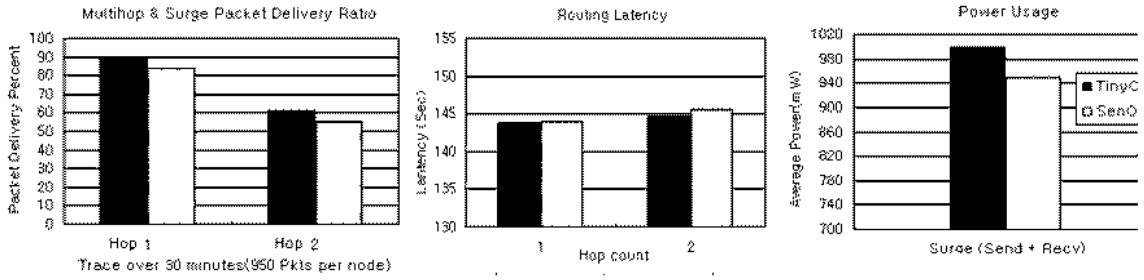
SenOS 전반적인 성능 평가는 현재 연구가 진행 중이고, 본 논문에서는 구현된 SenOS의 모듈별 수행시간을 측정하고 동일한 플랫폼에서 같은 응용 프로그램을 TinyOS와 SenOS 상에서 실행시킨 이후 결과를 비교하여 정리하였다.

4.1 SenOS 모듈별 수행시간

구현된 SenOS의 모듈별 수행시간은 간단하게 구성된 응용 프로그램을 이용하였고 파라미터를 달리하여 (상태 전이 테이블의 크기, 메모리의 크기 등) 측정하였다. senosSequencer는 이벤트 큐를 검사하고 발생한 이벤트를 상태 전이 테이블과 비교하여 연관된 콜백 함수를 호출하는 함수로써 수행 시간은 이벤트 큐에서 이벤트를 가져오는 시간과 상태 전이 테이블 검색 시간, 호출되는 콜백 함수의 수행 시간을 합하여 계산되어야 한다. 따라서 콜백 함수의 수행 시간을 줄이는 것이 시스템의 반응성을 향상시키는데 큰 역할을 하고 콜백함수 설계시 중요한 요소가 된다. 이러한 과정으로 나머지 모듈들의 수행 시간을 측정하였으며 그

Current State	Input event	Destination State	Action
Idle	recv packet	Checkmsg	check recv packet, create surge or mhop msg
Checkmsg	recv surge msg	Recvsurge	update neighbor info, upcate routing table or select route
Checkmsg	recv mhop msg	Recvmhop	update neighbor info, upcate routing table or create new entry
Recvsurge	ready data	Idle	select route, send recved surge packet
Recvmhop	ready data	Idle	do nothing
Idle	sense photo	Photo	start photo sense
Photo	complete photo sense	Idle	create 'sense humi' event
Idle	sense humi	Humi	start humi sense
Humi	complete humi sense	Idle	create 'sense temp' event
Idle	sense temp	Temp	start temp sense
Temp	complete temp sense	Idle	create 'route surge' event
Idle	route surge	Routesurge	get sensing data, select route, send surge packet
Routesurge	send complete	Idle	do nothing
Idle	send mhop	Routemhop	choose parent, update routing table, send mhop packet
Routemhop	send complete	Idle	do nothing

<그림 5> Surge 응용 프로그램용 상태 전이 테이블



<그림 6> 멀티 홉과 서지 패킷 전달 비율, 라우팅 지연 시간, 파워소모

결과는 가장 복잡한 모듈인 상태전이표변경모듈(senosChangeSTT)이 104.25μs 이고, 이벤트를 처리 모듈은 2.5μs의 수행속도로 측정되었다. 모듈별 코드 사이즈는 200byte 미만으로서 메모리 제한적인 센서 노드에 적합하며 모듈별 수행시간도 큰 편차없이 균등한 편이다. senosChangeSTT() 같은 함수는 동적 재구성시 수행되는 것이므로 전체적으로 무난한 처리 속도를 보여주고 있다.

4.2 TinyBench를 이용한 평가

SenOS의 성능평가는 대표적인 센서 네트워크 운영체제인 TinyOS와의 비교를 통해 측정하였으며 TinyBench[12]라는 TinyOS 성능 평가에 사용되었던 환경으로 수행 하였다. TinyBench에서 성능 평가를 위해 사용했던 Surge 응용 프로그램을 운영체제만 변경하고 동일한 플랫폼에서 수행시켜 보았고 실험 환경은 다음과 같다.

(1) 표1의 센서 노드 모듈을 이용하여 실험
(2) 7개 센서 노드를 연구실 내에 약 2m~3m 간격으로 배치하고, RF power는 최소 파워인 -25dBm과 최대 파워인 0dBm으로 변경해가면서 측정 [17,18].
(3) 멀티 홉 메시지의 주기는 20초, 센싱의 주기는 2초로 설정

4.2.1 Surge

Surge 응용 프로그램은 멀티홉 라우팅 프로토콜을 사용한 응용 프로그램으로써 노드들은 주기적으로 센싱(조도, 온도, 습도)하고 멀티 홉 무선 링크를 통해 베이스 스테이션(이하 싱크 노드)으로 데이터를 전송(Surge Msg) 한다[11]. 각 노드는 이웃 노드들의 정보를 테이블로 유지하고 노드에서 생성된 데이터는 부모에게 전달되어 최종적으로 싱크 노드에 도달된다. 싱크 노드는 아이디가 0으로 설정되어 있으며 PC와 시리얼케이블로 연결되어 있는 싱크 노드는 다른 노드로부터 받은 정보를 출력하게 된다. Surge 노드들은 싱크 노드로부터 가장 적은 홉 카운트를 갖고 최적으로 평가된 링크 cost 값을 갖는 노드를 부모 노드로 선택하며 링크 cost 계산은 비콘 패킷(Multihop Msg)들을 주기적으로 브로드 캐스팅함으로써 수행한다. 이러한 Surge 응용 프로그램을 그림 5와 같이 SenOS에서 실행할 수 있도록 상태 모델링 한 후 상태 전이 테이블 생성하고 필요한 콜백 함수들을 추가하였다.

실험 결과는 그림 6에서 볼 수 있듯이 노드가 부모 노드를 발견하는데 걸리는 평균 시간은 TinyOS의 Surge와 거의 동일 하였다. 패킷 전송률은 TinyOS의 Surge가 더 좋게 측정되었다. 동일한 센서 노드를 이용하고, 동일한 작동환경에서 부가 기능이 전혀 없이 단순 패킷을 전송하는 실험에서 이와 같은 전송율의 차이가 난 것은 분명히 본 연구팀에서 구현한 시스템의 구현상 문제가 있는 것이지만, 이것은 운영체제 자체의 구현 효율성 때문이 아니라 RF 모듈에 대한 드라이버 구현상, 본 연구팀의 구현 기술의 부족 때문이라고 판단된다.

Tiny Surge와 Sen Surge 응용 프로그램이 소모하는 파워 측정은 1ms 단위로 샘플링 해서 수행 시간을 측정하고, 실험의 신뢰성을 향상시키기 위해서 Tiny Surge의 경우 Sen Surge에서 사용되지 않는 기능을 사용하는 컴포넌트가 있는데 이때 수행되는 처리 시간을 제외한 부분을 가지고 결과 도출하였다. 아래는 실험에 사용된 센서 노드의 마이크로컨트롤러가 Active 상태일 때 사용되는 ms당 전력량은 16.5mW이다[15]. 이 때 멀티 홉 메시지와 서지 메시지를 전송 및 수신 때 사용되는 평균 전력량 (라우팅 경로가 설정 될 때까지 보내지는 평균 시간인 140초 동안 소비되는 전력량)은 TinyOS의 경우 998.25mW이며 SenOS는 950.0mW였다. 이와 같이 TinyOS의 surge 응용 프로그램의 전력 소모가 더 많았는데 컴포넌트를 사용하는 TinyOS의 경우 컴포넌트 연결이 많아질 경우 클럭 사이클이 4인 call과 ret명령어가 많이 사용되고 결과적으로 전체적인 수행 사이클이 증가하는 경향을 보이기 때문이다. 또한 TinyOS의 컴포넌트는 어떤 동작에 대한 호출이 커맨드를 통해 실행되고 이에 대한 성공 여부를 이벤트를 통해 회신하는데 일반적으로 반응을 필요로 하는 함수나 하드웨어의 제어에는 이 방법이 적당하지만 컴포넌트 사용이 많아질 경우 상위 애플리케이션 개발을 복잡하게 만드는 단점이 있다. 본 실험에서 사용한 Surge 응용의 경우

이러한 컴포넌트 계층구조(component hierarchy)가 크지 않았지만, 실제 프로그램에서 응용프로그램의 계층구조가 보다 복잡해 질 경우†, 이러한 소비전력 차이는 더욱 벌어질 것으로 쉽게 예측할 수 있다.

4.2.2 스케줄링 오버헤드와 CPU 오버헤드

SenOS의 스케줄링 오버헤드와 CPU 오버헤드를 측정하기 위해 타이머 인터럽트와 Blank 함수(아무 일도 수행하지 않는 함수)를 이용하여 TinyOS의 스케줄링 오버헤드와 SenOS의 스케줄링 오버헤드를 비교하였다. 하나의 태스크(Blank 함수)가 실행될 때까지 수행되는 함수들을 추적하여 기계어로 변환되었을 때 수행되는 명령어 셋을 조사하여 평가하였다. 총 500ms 동안 시스템이 Blank함수를 스케줄링 하는데 사용하는 평균 명령어의 개수와 실행 사이클을 측정한 결과와 10ms 동안 시스템이 Sleep 상태가 될 때까지 CPU의 Active Time을 측정한 결과는 아래와 같다.

기본적인 스케줄링에 드는 비용과 운영체제 자체의 코드 크기는 SenOS가 조금 크지만 센서 노드의 메모리 보유량을 고려할 때 문제가 없는 수준이다. 메모리 요구량 역시 SenOS가 다소 크지만 센서 노드 HW 구성상 전혀 무리가 없는 수준임을 알 수 있다.

5. 결 론

기존의 센서 노드의 운영체제들은 응용 프로그램 개발 편의성, 동적 재구성 기능의 지원 여부 및 동적 재구성 기능을 지원하는데 따른 추가적인 비용들과 관련하여 해결해야 할 문제점들을 가지고 있었다. 이러한 문제들을 해결하기 위해 FSM 기반으로 설계된 SenOS를 제한된 메모리와 컴퓨팅 파워를 가지는 센서 노드의 시스템 자원을 효율적으로 사용하고 표준 API제공과 동적 재구성 기능을 지원하도록 구현하였다. 구현된 SenOS와 TinyOS를 비교하였을 때 코드 크기는 SenOS가 조금 크지만 센서 노드의 메모리 보유량을 감안할 때 문제가 없는 수준이다. 메모리 요구량 역시 SenOS가 다소 크지만 이것은 센서 네트워크 응용에 필수적인 동적 재구성성을 감안하고 소프트웨어 개발 도구들과의 연계사용을 통한 개발편의성 및 코드 안정성을 얻기 위해 필요한 최소한의 비용이며 센서 노드 하드웨어 구성상 전혀 무리가 없는 수준임을 알 수 있다. 중요한 것은 이와 같은 이점들이 코드 최소화를 목표로 한 TinyOS와 비교해 보아도 큰 차이가 없는 수준이라는 점이고 실제로 TinyOS에 동적 재구성성을 지원하기 위해 VM을 추가한 Mate는 이보다 훨씬 큰 크기를 갖게 되며 동작 속도상 현저한 성능저하를 보이게 된다.

SenOS는 상태 전이 테이블의 교체와 콜백 함수의 변경을 이용하여 동적 재구성 기능을 지원하게 된다. 센서 노드의 응용 프로그램 변경시 필요한 콜백 함수가 없는 경우 새로운 콜백 함수를 노드로 내려보내야 하기 때문에 갱신 비용 절감에 대한 기대치는 낮아지겠지만 콜백 함수가 시스템에 있는 경우 상태 전이 테이블 교체만으로도 업데이트가 가능하게 되어 업데이트 비용이 SOS의 모듈 업데이트 비용보다 작아 질 수 있다. David Harel에 따르면 센서네트워크 응용과 같은 reactive distributed system에서 상태기반 모델링의 유용성은 입증된 바 있다[22]. 즉, FSM 기반의 새로운 실행모델은 범용 운용체제의 기능을 요구하지 않는 센서 노드용 응용 프로그램을 위한 최적의 구조임을 주장하는 것은 논리적으로 타당하다. 향후 SenOS의 동적 재구성 기능의 효율성 평가를 위해 TinyOS의 경우와 비교하고 콜백함수 변경의 효율성을 SOS와 비교, 평가함으로써 FSM 기반 운영체제의 센서노드에서의 유용성을 확실하게 입증할 예정이다.

† 컴포넌트 기반의 nesC 프로그램 모델상 복잡한 응용에서 컴포넌트 계층성 증가는 매우 자연스러운 설계 결과이다.

참 고 문 헌

[1] I. F. Akyildiz, W. Su et al., A Survey on Sensor Networks. IEEE Communications Magazine, Aug., 2002.

[2] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, System Architecture Directions for Networked Sensors. International Conference on Architectural Support for

Programming Languages and Operating Systems (2000).

[3] C.-C. Han, R. Rengaswamy, R. Shea, E. Kohler, and M. Srivastava. Sos: A dynamic operating system for sensor networks. In MobiSYS '05: Proceedings of the 3rd internatio-nal conference on Mobile systems, applications, and services. ACM Press, 2005.

[4] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, R. Han. MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platform. Mobile Networks & Applications (MONET) Journal, Special Issue on Wireless SensorNetworks, August 2005.

[5] Job Mulder, Stefan Dulman, Lodewijk van Hoesel, and Paul Havinga. PEEROS - System Software for Wireless Sensor Networks, August 2003.

[6] T.-H. Kim, Seongsoo Hong, State machine based operating system architecture for wireless sensor networks. Lecture Notes in Computer Science, Vol. 3320 No. pp. 803-803, Dec. 2004.

[7] P. Levis and D. Culler. Mate: A tiny virtual machine for sensor networks. In International Conference on Architectural Support for Programming Languages and Operating Systems. San Jose, CA, USA, Oct. 2002.

[8] Crossbow Technology, Inc. Mote In Network Programming User Reference, 2003

[9] A. Dunkels, B. Grönvall, T. Voigt. Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. In Proceedings of the First IEEE Workshop on Embedded Networked Sensors, Tampa, Florida, USA, November 2004.

[10] A. Dunkels, N. Finne, J. Eriksson, T. Voigt. Run-Time Dynamic Linking for Reprogramming Wireless Sensor Networks. In Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems, Boulder, Colorado, USA, November 2006.

[11] A. Woo, T. Tong, and D. Culler. Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. In First ACM international conference on Embedded Networked Sensor Systems, Los Angeles, CA, USA, Nov. 2003.

[12] M. Hempstead, M. Welsh, D. Brooks, TinyBench: The Case For a standardized Benchmark uite for TinyOS Based Wireless Sensor Network Devices. pp. 585-586, 29th Annual IEEE International Conference on Local Computer Networks.

[13] J. Labrosse, MicroC/OS-II The Real-Time Kernel 2nd Ed., CMP BOOKS

[14] M. Samek, Practical Statecharts in C/C++. CMP BOOKS

[15] TIS Committee. Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification Ver. 1.2, May 2003.

[16] C. Intanagonwiwat, R. Govindan, D.Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking, pp. 56-57, Boston, MA, USA, Aug. 2000.

[17] AVR128L Processor Data sheet, http://www.atmel.com

[18] Chipcon's CC2420 Data sheet, http://www.chipcon.com

[19] TinyOS Application, http://www.tinyos.net

[20] P. Jansen, S. Mullender, P. Havinga, J. Scholten: Lightweight EDF scheduling with deadline inheritance. Technical report (TR-CTIT -03-23), Centre for Telematics and Information Technology, University of Twente (2003)

[21] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, D. Culler, The nesC Language: A Holistic Approach to Networked Embedded

Systems. In Proceedings of PLDI, June 2003.
[22] D. Harel, Statecharts: A Visual Formalism for Complex Systems, The Science of Computer Programming, pp. 231-274, 1987
[23] 류정탁, 차부삼, 김연보, 문병현, 산업용 무선온도 측정 시스템 개발. 한국산업정보학회 논문지, Vol. 14, No. 1, 2009년 3월.
[24] 박홍진, 유비쿼터스 센서 네트워크를 이용한 독거노인 지킴이 시스템 구현. 한국산업정보학회 논문지, Vol. 15, No. 2, 2010년 6월.

하 승 현 (Seung Hyun Ha)

- 한양대학교 컴퓨터공학과 공학사
- 한양대학교 컴퓨터공학과 공학석사
- 삼성전자 무선사업부, System SW 엔지니어
- 관심분야 : 무선네트워크, 센서네트워크, 시스템 소프트웨어

김 태 형 (Tae-Hyung Kim)

- 정회원
- 서울대학교 컴퓨터공학과 공학사
- 서울대학교 컴퓨터공학과 공학석사
- 메릴랜드주립대학교 컴퓨터학과 컴퓨터학 박사
- 한양대학교 공학대학 컴퓨터공학과 부교수
- 관심분야 : 소프트웨어구조, 미들웨어 시스템, 무선 센서네트워크, 임베디드 시스템

논 문 접 수 일 : 2011년 03월 25일
1차수정완료일 : 2011년 05월 19일
게 재 확 정 일 : 2011년 05월 25일