확장성 및 실시간성을 고려한 실시간 센서 노드 플랫폼의 설계 및 구현

정회원 정 경 훈*, 준회원 김 병 훈**, 이 동 건**, 정회원 김 창 수***, 종신회원 탁 성 우****°

Design and Implementation of a Scalable Real-Time Sensor Node Platform

Kyung-hoon Jung* ***Regular Member***, Byoung-hoon Kim**, Dong-geon Lee** ***Associate Members***, Chang-soo Kim*** ***Regular Member***, Sung-woo Tak**** ***Lifelong Member***

요 약

본 논문에서는 멀티 태스크 기반의 확장성과 주기 및 비주기 태스크 관리 기법을 효율적으로 제공할 수 있는 실시간 센서 노드 플랫폼을 설계하고 구현하였다. 기존의 센서 네트워크 운영체제는 주기 및 비주기 태스크간의 효율적인 스케줄링 기법을 제공하지 않기 때문에 우선순위가 높은 비주기 태스크의 실행 선점으로 인해 주기 태스크의 마감시한을 보장할 수 없다. 이에 본 논문에서 제안한 주기 및 비주기 태스크 관리 기법은 운영체제 수준에서 주기 태스크의 마감시한 보장과 더불어 비주기 태스크의 평균 응답시간을 최소화할 수 있다. 또한 센서 노드 플랫폼에 용이한 확장성을 제공하기 위하여 멀티 태스크 기반의 동적 컴포넌트 실행 환경이 보장되는 센서 노드 플랫폼을 초경량 8비트 마이크로프로세서인 Atmel사의 Atmega128L이 탑재된 센서 보드에서 구현하였다. 구현된 실시간 센서 노드 플랫폼의 동작을 시험한 결과, 주기 태스크의 마감시한 보장을 제공함과 동시에 향상된 비주기 태스크의 평균 응답시간과 효율적인 시스템의 평균 처리기 이용률을 확인할 수 있었다.

Key Words : Sensor Networks, TCP/IP, Ubiquitous Computing

ABSTRACT

In this paper, we propose a real-time sensor node platform that guarantees the real-time scheduling of periodic and aperiodic tasks through a multitask-based software decomposition technique. Since existing sensor networking operation systems available in literature are not capable of supporting the real-time scheduling of periodic and aperiodic tasks, the preemption of aperiodic task with high priority can block periodic tasks, and so periodic tasks are likely to miss their deadlines. This paper presents a comprehensive evaluation of how to structure periodic or aperiodic task decomposition in real-time sensor-networking platforms as regard to guaranteeing the deadlines of all the periodic tasks and aiming to providing aperiodic tasks with average good response time. A case study based on real system experiments is conducted to illustrate the application and efficiency of the multitask-based dynamic component execution environment in the sensor node equipped with a low-power 8-bit microcontroller, an IEEE802.15.4 compliant 2.4GHz RF transceiver, and several sensors. It shows that our periodic and aperiodic task decomposition technique yields efficient performance in terms of three significant, objective goals: deadline miss ratio of periodic tasks, average response time of aperiodic tasks, and processor utilization of periodic and aperiodic tasks.

* 부산대학교 U-Port정보기술산학공동사업단 (jungkh@pknu.ac.kr), ** 부산대학교 정보컴퓨터공학부
*** 부경대학교 전자컴퓨터정보통신공학부 (cskim@pknu.ac.kr)
**** 부산대학교 정보컴퓨터공학부/컴퓨터 및 정보통신연구소, (swtak@pusan.ac.kr)(°:교신저자)
논문번호 : KICS2007-02-001, 접수일자 : 2007년 2월 16일, 최종논문접수일자 : 2007년 7월 13일

Ⅰ. 서 론

유비쿼터스 센서 네트워크 환경에서의 초소형 센서 노드는 PC나 PDA에 비해 매우 낮은 하드웨어 사양을 가지고 있음에도 불구하고 센싱 및 센싱된 데이터의 실시간 처리 작업, 그리고 노드간의 데이터 송수신 작업을 동시에 병행(concurrency)해야 한다. 따라서 센서 노드에서는 효율적인 다중 작업의 제어 및 관리와 실시간성을 제공할 수 있는 경량 운영체제가 필요하다. 1990년대 말부터 시작된 경량 센서 네트워크 운영체제에 관한 연구는 제한된 하드웨어 자원을 효율적으로 사용할 수 있는 기법에 대하여 초점을 두었다. 현재 TinyOS[1-2], MANTIS[3], DCOS[4], 그리고 AvrX[5]와 같은 다양한 경량 운영체제가 개발되고 있으며, 이러한 센서 네트워크 운영체제들은 공기 및 물의 오염 측정과 같은 환경오염 모니터링, 집안의 조명 및 창문의 원격 제어와 같은 디지털 홈, 그리고 군사 감시와 같은 실시간성이 요구되는 분야에 다양하게 적용되고 있다. 그러나 기존의 센서 네트워크 운영체제는 실시간 주기 및 비주기 태스크의 실행을 동시에 고려하는 스케줄링 기법을 제공하지 않기 때문에 우선순위가 높은 비주기 태스크의 실행 선점으로 인해 주기 태스크의 마감시한을 보장할 수 없다. 따라서 실시간 환경에서 적용될 센서 네트워크 운영체제는 제한된 하드웨어 자원의 효율적인 관리뿐만 아니라 요청된 작업의 결정성 및 신뢰성을 제공할 수 있어야 한다. 요청된 작업의 결정성은 명시된 시간 내에 요청된 서비스의 실행 가능 및 완료에 대한 보장을 의미하며, 신뢰성은 다른 작업의 실행 오류와 상관없이 요청된 서비스의 제공 가능성을 의미한다. 일반적으로 실시간 시스템에서의 태스크 집합은 실행시간 간격이 일정한 주기 태스크와 그렇지 않은 비주기 태스크로 구성된다. 주기 태스크는 마감시한(deadline) 내에 실행완료가 보장되어야 하며, 우선순위가 높은 비주기 태스크의 실행 선점으로 인해 주기 태스크의 마감시한을 보장할 수 없는 경우가 발생하지 않아야 한다. 이에 운영체제 수준에서 주기 태스크의 마감시한 보장과 더불어 비주기 태스크의 평균 응답시간을 최소화할 수 있는 기법이 필요하다. 그러나 기존의 센서 네트워크 운영체제인 TinyOS는 실시간성을 지원하지 못하며, MANTIS와 DCOS는 주기 태스크들에 대한 스케줄링만 지원한다. 이와 같은 운영체제 환경에서 주기 태스크의 마감시한 보장과 비주기 태스크의 빠른 응답시간이 요구되는 응용 서비스를 제공하기 위해서는 개발자가 응용 계층에서 별도의 실시간 스케줄링 기법을 설계 및 구현해야 하는 문제점이 발생한다. 이러한 문제점은 빠른 개발 공정주기를 요구하는 센서 네트워크 산업에서 다양한 실시간 응용 서비스의 개발 및 관련 제품의 출시를 지연시킨다. 이에 본 논문에서는 운영체제 수준에서 주기 태스크의 마감시한 보장뿐만 아니라 비주기 태스크의 빠른 평균 응답시간과 효율적인 시스템의 평균 처리기 이용률(Average Processor Utilization)을 제공할 수 있는 실시간 센서 네트워크를 위한 태스크 관리 기법을 제안한다. 또한 센서 노드 시스템에 용이한 확장성을 제공하기 위하여 멀티 태스크 기반의 동적 컴포넌트 실행 환경이 보장되는 센서 노드 플랫폼을 초경량 8비트 마이크로프로세서인 Atmel사의 Atmega128L이 탑재된 센서 보드에서 구현하여 성능을 검증한다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 관련 연구를 기술하였고, Ⅲ장에서는 확장성을 고려한 실시간 센서 노드 플랫폼의 구성과 주기 및 비주기 태스크간의 실시간 관리 기법에 대해 기술하였다. Ⅳ장에서는 제안된 실시간 센서 노드 플랫폼의 성능평가를 기술하였다. 마지막으로 Ⅴ장에서는 결론을 기술하였다.

Ⅱ. 관련연구

이 장에서는 기존의 센서 네트워크 운영체제와 임베디드 시스템 운영체제에서 고려하고 있는 실시간 스케줄링 기법에 대하여 기술한다. 컴포넌트 기반의 센서 네트워크 운영체제인 TinyOS에서는 서비스의 추가 및 삭제가 용이하여 응용 서비스의 빠른 개발이 가능하다[1]. 그러나 TinyOS는 FIFO (First-In First-Out) 큐 방식의 비선점형 태스크 스케줄링만 제공하기 때문에 즉시 실행이 필요한 태스크임에도 불구하고 자신의 우선순위보다 낮은 태스크가 획득한 CPU 사용권한을 선점할 수 없다. 앞에서 언급한 TinyOS에서의 FIFO기반 비선점형 태스크 대기 큐의 크기는 7로 설정되어 있다. 따라서 TinyOS기반의 센서 노드 시스템에서는 큐에 도착하는 태스크의 실행 요구 비율보다 태스크의 처리율이 작을 경우, 큐의 오버플로가 발생하여 이후에 도착하는 태스크들을 처리할 수 없는 시스템 과부하(overload) 상태가 발생된다[2]. 2006년에 개발 완료된 TinyOS 2.x 버전에서의 스케줄링은

개발자의 서비스 추가 및 삭제가 용이한 컴포넌트 형태로 제공된다. 또한 비선점형 기반의 EDF (Earliest Deadline First)[6] 스케줄링을 제공하여 부분적인 실시간 서비스를 지원할 수 있다. 그러나 비선점형 스케줄링 기반에서 주기 및 비주기 태스크의 스케줄링 가능성을 제공하는 방법은 NP-hard 문제로 알려져 있다[7-8]. AvrX와 MANTIS는 비선점형 운영체제인 TinyOS에서 실행되는 태스크에게 선점형 실행을 제공하기 위하여 TinyOS 자체를 하나의 쓰레드로 취급하여 실행시킨다. EDF 기반의 선점형 실시간 센서 네트워크 운영체제인 DCOS에서는 센싱된 데이터 및 브로드캐스팅으로 수신된 데이터를 효율적으로 처리할 수 있는 데이터 중심 아키텍처(data centric architecture) 개념을 사용한다. 그러나 EDF는 주기 태스크만을 고려할 경우 실시간 스케줄링을 보장할 수 있지만, 주기 태스크의 마감시한 보장과 함께 비주기 태스크의 빠른 평균 응답시간의 보장까지는 제공하지 못한다. 한편, 일반적으로 많이 사용되고 있는 임베디드 운영체제를 살펴보면 다음과 같다. VxWorks는 주기 태스크의 생성 및 실행을 위한 API (Application Programming Interface)를 제공하지만 생성된 주기 태스크의 마감시한 보장은 사용자에 의해 구현되어야 한다[9]. 또한 VxWorks의 커널 소스는 비공개이므로 사용자가 커널의 직접적인 수정을 통하여 주기 태스크의 마감시한 보장 및 비주기 태스크의 빠른 응답시간을 보장하는 것은 불가능 하다. RT-Linux는 비선점형 리눅스 커널과 선점형 실시간 커널이 함께 공존하는 이중 커널 구조를 가지고 있으며, 실시간 태스크는 실시간 커널에 의해 관리된다[9]. 비실시간 태스크와 실시간 태스크 간의 데이터 송수신은 RT_FIFO 큐를 통해 이루어진다. RT_FIFO 큐를 통해 비실시간 태스크가 실시간 태스크를 종료시킬 수도 있기 때문에 실시간 태스크의 결정성을 보장할 수 없는 경우가 발생할 수 있다. 그리고 주기 및 비주기 태스크들은 우선순위에 의해 스케줄되기 때문에 높은 우선순위와 긴 실행시간을 가지는 비주기 태스크의 실행으로 인해 보다 낮은 우선순위를 가진 주기 태스크의 실행과 마감시한을 보장할 수 없게 된다. 경량 실시간 임베디드 운영체제인 uC/OS-II[10]에서는 태스크들이 우선순위에 따라 스케줄된다. *uC/OS-II*는 실시간 임베디드 시스템을 위해 개발된 운영체제이지만, 주기 태스크의 마감시한 보장 및 비주기 태스크의 빠른 응답시간을 보장하는 자료구조와 기법을 제공하지 않는다.

Ⅲ. 실시간 태스크 관리 기법

이 장에서는 멀티 태스크 기반 서비스 컴포넌트의 확장성과 주기 및 비주기 태스크 관리 기법을 효율적으로 제공할 수 있는 실시간 센서 노드 플랫폼을 기술하였다. 그림 1에서 보는 바와 같이 본 논문에서 제안하는 실시간 센서 노드 플랫폼은 확장성 및 주기 태스크의 마감시한을 보장하기 위하여 응용 및 시스템 소프트웨어 컴포넌트를 태스크 단위로 실행한다.

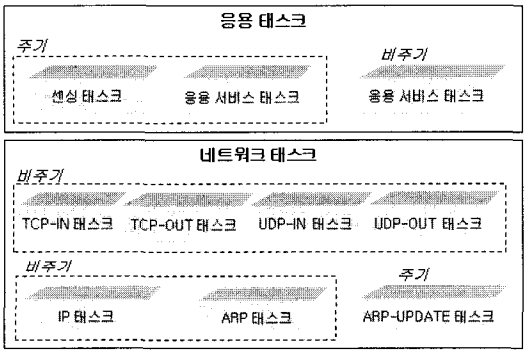

그림 1. 실시간 센서 노드 플랫폼에서 실행되는 태스크의 구성

본 논문에서는 응용 서비스 컴포넌트와 네트워크 컴포넌트를 주기 및 비주기 태스크로 구현하였다. 응용 태스크의 구성을 살펴보면 다음과 같다. 센싱 태스크는 일정한 시간 간격으로 센서를 구동시켜 센싱 데이터를 수집하며, 수집된 데이터를 네트워크 태스크에게 전달한다. 주기 및 비주기 응용 태스크는 사용자가 정의한 응용 서비스를 수행한다. 한편, 네트워크 컴포넌트의 구성을 위하여 각각의 네트워크 계층에 대응하는 프로토콜의 기능은 개별 태스크로 구성된다. 본 논문에서는 TCP/IP 프로토콜 스택을 태스크 단위로 설계 및 구현하였다. TCP/IP 프로토콜 스택은 제한적인 하드웨어 자원을 가진 센서 네트워킹 플랫폼을 위해 설계되지 않았으나 이미 검증된 계층형 통신 구조로서, 센서 네트워킹 플랫폼에서 센서 네트워크와 인터넷간의 연동을 담당하는 싱크 노드는 감지된 센싱 데이터 및 제어 정보를 전달하기 위하여 TCP/IP 프로토콜 스택을 사용해야 한다. 그리고 현재까지 센서 네트워킹을 담당하는 통신 프로토콜의 표준화가 아직 이루어지 않았지만, 제한된 하드웨어 자원을 가진

센서 노드에 탑재될 소프트웨어 통신구조는 기존의 TCP/IP 프로토콜보다 훨씬 단순해질 것이다. 따라서 본 논문에서 제안된 멀티 태스크 기반의 컴포넌트 확장성과 주기 및 비주기 실시간 태스크 관리 기법이 TCP/IP 프로토콜을 사용한 센서 네트워킹 환경에서 좋은 성능을 제공할 수 있다면, 향후 표준화될 경량 센서 통신 구조를 탑재한 센서 노드에서도 좋은 성능을 제공할 수 있을 것이다. 그림 1에서 보는 바와 같이 실시간 센서 노드 플랫폼에서 실행되는 TCP/IP 프로토콜 스택은 ARP 태스크, ARP-UPDATE 태스크, IP 태스크, UDP -IN 태스크, UDP-OUT 태스크, TCP-IN 태스크, 그리고 TCP-OUT 태스크로 분류된다. TCP/IP 프로토콜 스택에서 주기 및 비주적인 실행 속성을 가진 네트워크 프로토콜의 기능을 살펴보면 다음과 같다. 먼저 ARP 프로토콜에서 20분마다 주기적으로 ARP 캐쉬 테이블을 갱신하는 기능과 TCP 프로토콜에서 주기적으로 실행되는 TCP 타이머가 있다. ARP 캐쉬 테이블을 갱신하는 기능은 ARP-UPDATE 태스크가 담당한다. 그리고 TCP 타이머의 기능을 담당하는 실행 코드의 시간이 태스크의 문맥교환 시간보다 훨씬 짧기 때문에, 본 논문에서 TCP 타이머는 일정한 실행주기를 가지는 태스크가 아닌 타이머 인터럽트 서비스 루틴 내에 실행되도록 설계 및 구현하였다. IP 및 UDP를 포함한 나머지 ARP와 TCP 및 UDP 프로토콜의 기능은 일정한 주기로 실행되지 않기 때문에 다음과 같이 비주기 태스크로 설계 및 구현하였다. 먼저 ARP 프로토콜의 메시지 처리를 담당하는 비주기 ARP 태스크와 IP 패킷의 처리를 담당하는 비주기 IP 태스크가 있다. TCP 타이머를 제외한 나머지 TCP 프로토콜의 기능은 다중 통신 채널을 통해 TCP 패킷의 수신과 송신을 담당하는 TCP-IN 태스크와 TCP-OUT 태스크가 담당한다. UDP 프로토콜의 기능은 다중 통신채널을 통해 UDP 패킷의 수신과 송신을 담당하는 UDP-IN 태스크와 UDP-OUT 태스크가 담당한다. 그림 2는 센서 노드 플랫폼에서 실행되는 주기 및 비주기 태스크의 실시간 관리를 위한 시스템의 구성을 보여주고 있다. 이 시스템은 주기 태스크, 주기 및 비주기 태스크 스케줄러, 그리고 비주기 태스크로 구성되어 있다. 이 시스템의 동작 구성을 살펴보면 다음과 같다. 주기 태스크는 센싱 태스크 및 응용 서비스 태스크, 그리고 ARP-UPDATE 태스크로 구성된다. 주기 및 비주기 태스크 스케줄러는 스케줄 가능성 검사자 모듈 및 여유시간 관리 모듈, 그리고 비주기 태스크 스케줄러 모듈과 주기 태스크 스케줄러 모듈로 구성되어 있다. 주기 및 비주기 태스크 스케줄러를 구성하고 있는 각 세부 모듈의 기능을 살펴보면 다음과 같다. 스케줄 가능성 검사자 모듈은 주기 태스크의 스케줄 가능성 여부를 검사한다. 여유시간 관리 모듈은 비주기 태스크의 실행 시간 할당과 주기 태스크의 처리 여유시간을 계산하며 여유시간 계산자 모듈과 여유시간 할당자 모듈로 구성되어 있다. 여유시간 계산자 모듈은 비주기 태스크가 실행되는 여유시간을 계산하여 여유시간 테이블에 저장한다. 여유시간 할당자 모듈은 비주기 태스크 스케줄러 모듈이 선택한 비주기

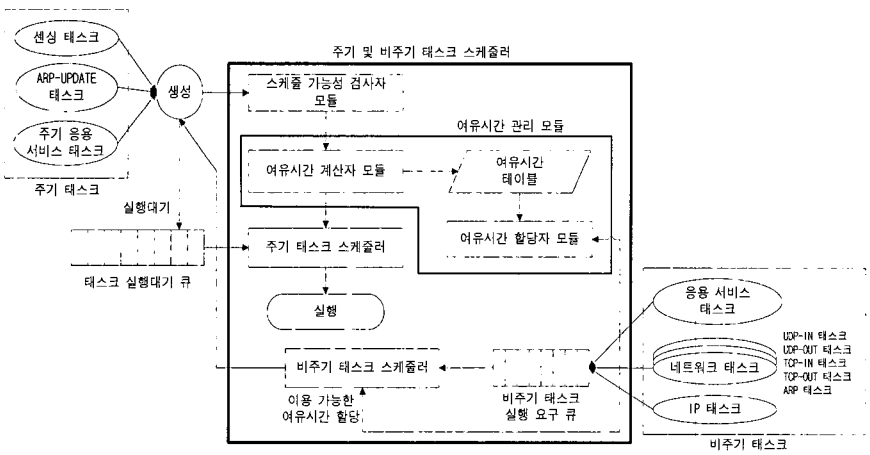

그림 2. 실시간 태스크 관리를 위한 시스템의 구성

태스크에게 여유시간을 할당한다. 비주기 태스크 스케줄러 모듈은 여유시간 관리 모듈이 할당한 여유시간을 가장 먼저 실행이 필요한 비주기 태스크에게 제공한다. 마지막으로 주기 태스크 스케줄러 모듈에서 주기 태스크와 비주기 태스크 중에서 가장 높은 우선순위의 태스크가 선택된 후 실행된다. 본 논문에서는 주기 태스크의 스케줄링을 위해 RM(Rate Monotonic) 알고리즘을 사용하였으며, 비주기 태스크를 위해 Slack Stealing 알고리즘을 사용하였다[11-12]. 다음 절에서는 스케줄 가능성 검사자 모듈 및 여유시간 관리 모듈, 그리고 비주기 태스크 스케줄러 모듈에서 수행하는 상세 처리 과정을 기술하였다. 먼저 스케줄 가능성 검사자 모듈의 세부 사항을 살펴보면 다음과 같다.

3.1 스케줄 가능성 검사자 모듈

스케줄 가능성 검사자 모듈은 주기 태스크의 스케줄 가능성 여부를 검사한다. 각 주기 태스크의 처리기 이용률을 기반으로 하여 전체 처리기 이용률을 계산한다. 전체 처리기 이용률의 계산을 위해 다음과 같은 용어를 사용한다. n개의 주기 태스크 집합 $\tau=\{\tau_1,\tau_2,...,\tau_n\}$가 주어질 때 각 태스크 τ_i의 도착시간과 실행시간, 마감시한, 실행 주기를 각각 R_i, C_i, D_i, T_i라고 하면, $\tau_i=(R_i, C_i, D_i, T_i)$로 표현된다. n개의 태스크로 구성된 주기 태스크 집합에 대해서 다음과 같은 가정을 한다. τ_i 는 τ_{i-1}보다 우선순위가 낮다. 즉, 우선순위 i를 가지 는 태스크 τ_i는 우선순위 i보다 큰 태스크들의 수행이 끝난 후에 태스크 실행이 가능하다. 주기 태스크 집합은 모든 태스크들의 초기 시작시간이 동일한 태스크 집합이며 시스템은 하나의 처리기를 가진다. 또한 시간은 양의 정수로 나타낸다. 수식 (1)부터 수식 (5)까지의 내용을 살펴보면 다음과 같다. $W_i(t)$는 시간 t에 대한 태스크 τ_i가 수행완료를 위해 요구되는 처리기의 시간요구량 으로서 수식 (1)에 의해 계산되며, 시간 t는 수식 (5)에서 구한 S_i의 요소 값이 된다. $L_i(t)$는 각 태스크 τ_i에 할당된 값을 사용하여 계산된 처리기 이용률이다. 그리고 L_i는 수식 (2)에서 계산된 처리기 이용률인 $L_i(t)$ 중에서 최소값을 사용하며, 이 최소값은 태스크 τ_i의 실제 처리기 이용률이 된다. L은 주기 태스크 집합의 전체 처리기 이용률 이며 수식 (3)에서 구한 τ_i의 처리기 이용률 중에서 최대값이 된다. $L \le 1$이면, 시스템내의 처리기 내에 과부하가 발생되지 않기 때문에 주기 태스크의 집합은 스케줄이 가능하다. 그렇지 않으면, 스케줄이 불가능하게 된다.

$$W_i(t) = \sum_{j=1}^{i} C_j \cdot \lceil t / T_j \rceil \quad (1)$$

$$L_i(t) = W_i(t)/t \quad (2)$$

$$L_i = \min_{\{0 < t \le T_i\}} L_i(t) \quad (3)$$

$$L = \max_{\{1 \le i \le n\}} L_i \quad (4)$$

$$t = S_i = \{k \cdot T_j \mid j = 1,...,i\,; k = 1,..., \lfloor T_i / T_j \rfloor\} \quad (5)$$

예를 들어, 주기 태스크 집합 τ={(0,2,5,5), (0,3,10,10), (0,4,40,40)}가 주어진다고 가정할 경우, 표 1은 주기 태스크 집합 τ에 대한 전체 처리기 이용률의 계산을 나타낸다. 표 1에서 보는 바와 같이 L_1=0.40, L_2=min(1.00, 0.70)=0.70, 그리고 L_3=min(1.80, 1.10, 1.06, 0.90, 0.92, 0.83, 0.85, 0.80)=0.80이 된다. 따라서 태스크 집합 τ의 전체 처리기 이용률은 L=max(0.40, 0.70, 0.80)=0.80이 된다. L<1이므로 주기 태스크 집합 τ는 스케줄이 가능하다.

표 1. 주기 태스크 집합에 대한 전체 처리기 이용률

i	$\tau_i(T_i,C_i)$	t	$W_i(t)$	$L_i(t)$	L_i	L
1	τ_1(5, 2)	5	C_1=2	0.40	*0.40*	
2	τ_2(10,3)	5	C_1+C_2=5	1.00		
		10	$2C_1+C_2$=7	0.70	*0.70*	
3	τ_3(40,4)	5	$C_1+C_2+C_3$=9	1.80		
		10	$2C_1+C_2+C_3$=11	1.10		
		15	$3C_1+2C_2+C_3$=16	1.06		
		20	$4C_1+2C_2+C_3$=18	0.90		
		25	$5C_1+3C_2+C_3$=23	0.92		
		30	$6C_1+3C_2+C_3$=25	0.83		
		35	$7C_1+4C_2+C_3$=30	0.85		
		40	$8C_1+4C_2+C_3$=32	0.80	*0.80*	*0.80*

3.2 여유시간 관리 모듈

여유시간 관리 모듈에서 처리기 여유시간을 계산하기 위한 과정은 다음과 같다. 주기 태스크 τ_i의 j번째 요구를 τ_{ij}라고 하면, τ_{ij}의 도착시간은 $(j\text{-}1)T_i$이고, 마감시한은 D_{ij}, 그리고 실행시간은

c_{ij}가 된다. $a_i(t)$는 시간 [0, t]와 $0 \le t \le D_{ij}$ 조건에서 i보다 크거나 같은 우선순위를 가지는 비주기 태스크들의 전체 실행시간이 된다. $P_i(t)$는 시간 [0, t]에서 우선순위가 i인 주기 태스크가 준비되는 시간이 된다. $I_i(t)$는 시간 [0, t]에서 우선순위 i보다 낮은 태스크들이 수행되거나 처리기가 유휴상태인 시간이 된다. 태스크들의 전체 처리기 이용시간인 $W_i(t)$는 아래의 수식 (6)에 의해 계산된다.

$$W_i(t) = a_i(t) + P_i(t) + I_i(t) \quad (6)$$

A_{ij}를 시간 $[0, C_{ij}]$에서 우선순위가 i보다 크거나 같은 비주기 태스크를 처리하기 위한 최대 여유시간이라고 하면, 수식 (7)의 조건이 성립된다. C_{ij}는 주기 태스크 τ_{ij}의 종료시간이 된다.

$$\min_{\{0 \le t \le D_{ij}\}}\{(A_{ij} + P_i(t))/t\} = 1 \quad (7)$$

$A_i(t) = A_{ij},\ C_{ij-1} \le t < C_{ij},\ j \ge 1$ 이라고 하고, $A^*(t)$를 최대 여유시간이라고 하면,

$$A^*(t) = \min_{\{1 \le i \le n\}} A_i(t),\ C_{ij-1} \le t < C_{ij} \quad (8)$$

가 된다. 예를 들어 주기 태스크 집합 τ={(0,2,5,5), (0,3,10,10), (0,4,40,40)}가 주어진다고 가정할 경우, 모든 태스크 τ_i에서 실행 주기 T_i의 최소공배수가 되는 하이퍼주기(Hyperperiod)는 40이 된다. 그림 3은 수식 (6)에서부터 수식 (8)을 통해 계산된 주기 태스크 집합 τ에 대한 여유시간 테이블을 나타낸다.

그림 3에서 명암부분은 태스크 실행으로 인해 처리기가 busy인 상태를 의미한다. 표 1로부터 τ의 전체 처리기 이용률은 80%이고 하이퍼주기는 40이므로 비주기 태스크의 실행을 위해 필요한 처리기 여유시간은 8(20%)임을 알 수 있다. 그림 3에서 비주기 태스크 AT가 18인 시점에 도착 하였다면 여유시간 관리 모듈은 최대 2의 여유시간을 할당할 수 있다.

3.3 비주기적 태스크 스케줄러 모듈

비주기 태스크 스케줄러 모듈은 여유시간 관리 모듈로부터 할당받은 여유시간 테이블의 값을 가장 먼저 실행이 필요한 비주기 태스크에게 제공한다. 예를 들어 주기 태스크 집합 τ={(0,2,5,5), (0,3,10,10), (0,4,40,40)}과 실행시간이 9인 비주기 태스크 AT의 우선순위는 τ_2보다는 작고 τ_3보다는 클 경우를 가정한다. 이런 경우 주기 태스크 집합 τ의 하이퍼주기는 40이 된다. 그림 4는 주기 태스크 집합 τ와 AT를 우선순위 기반의 스케줄링에 적용할 경우 생성되는 태스크 할당 테이블을 보여준다. 그림 4에서 보는 바와 같이 비주기 태스크 AT는 실행을 완료하지만, 태스크 τ_3는 비주기 태스크 AT의 실행 간섭으로 인해 41에서 실행을 완료하게 되어 마감시한을 초과하게 된다. 그림 5는 여유시간 관리 모듈에서 할당받은 여유시간을 사용하여 비주기 태스크 AT가 스케줄링 될 경우 생성되는 태스크 할당 테이블을 보여준다. 그림 5에서 보는 바와 같이 비주기 태스크 AT는 여유시간 관리 모듈로부터 여유시간을 할당받아야만 스케줄링이 가능하므로 태스크 τ_3보다 우선순위가 높더라도 태스크 τ_3의 실행이 완료된 후에 스케줄링이 된다. 다시 말하면, 그림 3에서 보는 바와 같이 여유시간 테이블에서 할당 가능한 가장 빠른 여유시간은 시간은 18이기 때문에 비주기 태스크 AT는 18에서 스케줄이 가능하다. 그림 5에서 보는 바와 같이 본 논문에서 제안한 주기 및 비주기 태스크 관리 기법은 비주기 태스크인 AT의 실행을 보장함과 동시에 모든 주기 태스크들의 실행을 하이퍼주기 내에 완료시킨다.

3.4 주기 태스크 스케줄러 모듈

주기 태스크의 스케줄링 기능을 제공하는 주기 태스크 스케줄러 모듈은 선점 가능한 우선순위 기반의 스케줄링을 제공한다. 주기 태스크들의 우선순위는 RM[11] 알고리즘에 의해 부여되며 비주기

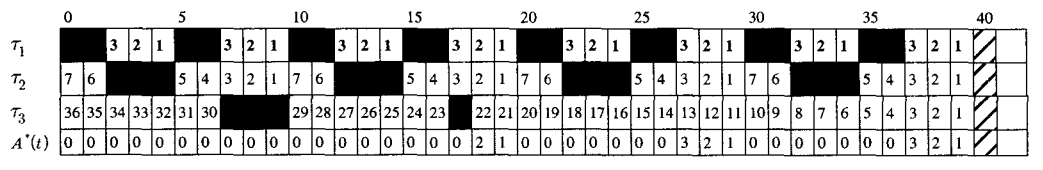

그림 3. 여유시간 매니저에 의해 생성된 여유시간 테이블

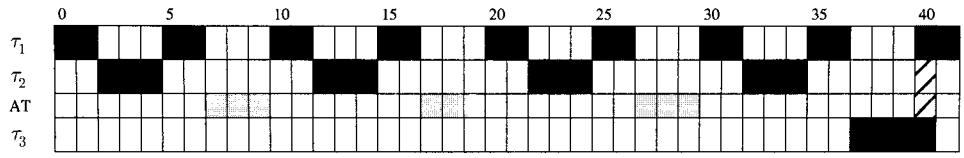

그림 4. 우선순위 기반에 의한 태스크 할당 테이블

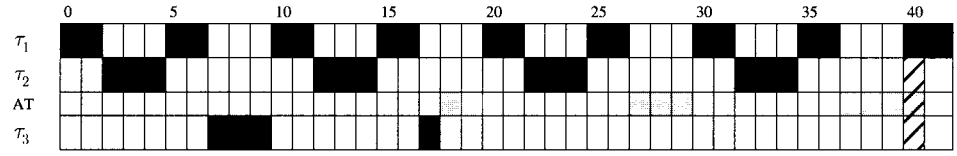

그림 5. 여유시간 관리 모듈 기반의 태스크 할당 테이블

태스크는 태스크의 중요도에 따라 우선순위가 임의로 정해질 수 있다. 주기 태스크 스케줄러 모듈은 주기 및 비주기 태스크의 우선순위 테이블에서 가장 높은 우선순위를 가지는 태스크를 선택하여 스케줄링을 한다.

Ⅳ. 비교 평가

본 논문에서 제안한 멀티 태스크 기반의 확장성과 주기 및 비주기 태스크 관리 기법을 효율적으로 제공할 수 있는 실시간 센서 노드 플랫폼은 다음과 같은 환경에서 구현 및 성능을 시험하였다. 실시간 센서 노드의 하드웨어 플랫폼은 chipcon사에서 개발한 CC2420DB 무선 센서 모트(Mote)를 사용하였다. CC2420DB 하드웨어 플랫폼은 IEEE802.15.4 표준을 지원하는 RF 송수신기(Transceiver)와 Atmel사의 Atmega128L 8비트 마이크로프로세서, 그리고 1개의 내부 센서 (온도 센서)가 있으며 동작시험을 위하여 추가적인 2개의 외부 센서(조도와 습도 센서)를 장착하였다. 그림 6에서 보는 바와 같이 실시간 센서 노드의 소프트웨어 플랫폼은 2.52 버전의 uC/OS-II와 0.90 버전의 uIP[13]를 기반으로 하여 3개의 성능 평가 모델을 구현하였다. uIP는 8-비트 마이크로프로세서에서 실행가능하도록 최적화된 경량 TCP/IP 프로토콜 스택 소프트웨어이다. 먼저 COSIP-I 모델은 기존의 uC/OS 운영체제와 uIP로 구성되어 있다. COSIP-II 모델은 기존의 uC/OS 운영체제 환경에서 그림 1에서 제시한 멀티 태스크 기반의 uIP로 구성되어 있다. 본 연구에서 기존의 uIP를 재수정한 멀티 태스크 기반의 uIP는 ARP 태스크, ARP-UPDATE 태스크, IP 태스크, UDP-IN 태스크, UDP-OUT 태스크, TCP-IN 태스크, 그리고 TCP-OUT 태스크로 구성되어 있다. 또한 새로운 프로토콜의 추가가 필요할 경우 태스크 형태로 구현될 수 있도록 태스크 기반의 컴포넌트 확장성을 제공한다. COSIP-III 모델은 본 논문에서 제안한 주기 및 비주기 태스크의 실시간 관리 기법이 적용된 uC/OS-II 운영체제와 멀티 태스크 기반의 uIP로 구성되어 있다. COSIP-I 모델에서는 기존 uIP 구조의 제약점인 다중 TCP/UDP 통신 연결을 제공하지 못한다. 그러나 COSIP-I 모델과는 다르게 COSIP-II 모델과 COSIP-III 모델은 다중 TCP/UDP 통신 연결이 가능하도록 설계되었다. 특히 COSIP-III 모델은 다른 두 모델과 다르게 운영체제 수준에서 주기 태스크의 마감시한 보장과 더불어 비주기 태스크의 평균 응답시간을 최소화할 수 있다. COSIP-I 모델과 COSIP-II 모델, 그리고 COSIP-III 모델의 성능을 평가하기 위하여 개별 IP가 할당된 2대의 CC2420DB 센서 모트를 이용하여 시험 환경을 구축한 후 동작 시험을 하였다. 동작 시험에서는 주기 태스크의 마감시한 보장 및 비주기 태스크의 평균 응답시간, 그리고 주기 및 비주기 태스크의 평균 처리기 이용률에 대한 성능을 측정하고 평가하였다. 표 2는 성능 평가를 위해 동작 시험에서 사용된 응용 태스크와 네트워크 태스크를 보여주고 있다. 응용 태스크는 센싱을 담당하는 3개의 태스크 (온도, 조도, 습도 측정 태스크)와 1개의 BIG-AP(BIG APeriodic) 태스크로 구성되어 있다. 비주기 BIG-AP 태스크는 200ms의 실행시간을 가진다. 그리고 비주기 BIG-AP 태스크는 모든 주기 태스크의 마감시한과 나머지 비주기

표 2. 성능 평가를 위한 태스크들의 상세 정보

태스크 이름		우선순위	주기 (ms)	실행시간 (ms)		
				COSIP-I 모델	COSIP-II 모델	COSIP-III 모델
응용 태스크	온도 측정 태스크	16	200	40	40	40
	조도 측정 태스크	17	300	10	10	10
	습도 측정 태스크	18	400	10	10	10
	BIG-AP 태스크	10	비주기	적용되지 않음	20	20
네트워크 태스크	ARP 태스크	9	비주기	10	10	10
	IP 태스크	10	비주기	2.7	2.7	2.8
	TCP-OUT 태스크	11	비주기	3.1	3.3	3.3
	TCP-IN 태스크	12	비주기	3.0	3.0	3.0
	UDP-OUT 태스크	13	비주기	3.0	3.0	3.0
	UDP-IN 태스크	14	비주기	3.0	3.0	3.0
	ARP-UPDATE 태스크	19	500	10	10	10

태스크의 응답시간을 계속적으로 지연시키기 위해서 평균 300ms마다 실행이 된다. COSIP-I 모델에서는 주기 및 비주기 태스크의 실시간 관리 기능이 없는 기존의 uC/OS-II를 활용하기 때문에 비주기 BIG-AP 태스크는 실험 환경에 포함되지 않는다. 온도 측정 태스크는 200ms마다 주기적으로 온도를 측정하며, 임계 온도를 초과하면 LED를 깜빡거린다. 조도 측정 태스크는 300ms마다 주기적으로 현재의 조도를 감지한 후 TCP 프로토콜 및 IEEE 802.15.4기반 무선 RF 통신을 이용하여 원격지에 있는 CC2420DB 모트에 센싱된 조도 데이터를 전송한다. 습도 측정 태스크는 400ms마다 주기적으로 현재 습도를 감지한 후 UDP 프로토콜 및 IEEE802.15.4기반 무선 RF 통신을 이용하여 원격지에 있는 CC2420DB 모트에 센싱된 습도 데이터를 전송한다. 센싱 데이터의 프레임 구조는 1byte의 ID 항목과 4바이트의 데이터 항목을 가진다. 그리고 조도 측정 태스크가 TCP 프로토콜을 사용하여 센싱된 데이터 프레임을 전송하는데 필요한 총 패킷의 크기는 40바이트 크기의 TCP/IP 헤더와 5바이트 크기의 센싱 데이터 프레임의 합인 45바이트가 된다. 습도 측정 태스크는 UDP 프로토콜을 사용하여 센싱된 데이터 프레임을 전송 하는데 필요한 총 패킷의 크기는 28바이트 크기의 UDP/IP 헤더와 5바이트 크기의 센싱 데이터 프레임의 합인 33바이트가 된다. 한편, 표 2에서 제시된 각 태스크의 우선순위는 다음과 같은 규칙에 의해 할당되었다.

1. 센서 네트워킹 플랫폼에서 무선 센서 모트가 감지 및 제어하고자 하는 데이터는 일정한 end-to-end 지연 시간 내에 교환되어야 의미 있는 정보가 되기 때문에, 데이터의 송수신을 담당하는 네트워크 태스크의 우선순위가 응용 태스크들보다 높다.
2. 응용 태스크간의 우선순위는 해당 응용내용의 중요도에 따라 결정된다.
3. 비주기 BIG-AP 태스크가 주기 태스크인 온도, 조도, 습도 측정 태스크의 마감시한과 나머지 비주기 태스크의 응답시간에 얼마나 심각한 영향을 주는지를 확인하기 위해서 다른 응용 태스크보다 높은 우선순위를 할당한다.
4. 본 논문에서 가정한 네트워크 태스크간의 우선순위 할당 규칙은 다음과 같다.
   - 4-A. ARP 캐쉬 테이블을 주기적으로 갱신하는 ARP-UPDATE 태스크의 기능은 TCP/IP 프로토콜에서 선택적인 기능이기 때문에 가장 낮은 우선순위를 할당한다.
   - 4-B. 상위 계층 프로토콜의 실행이 완료되기 위해서는 하위 계층 프로토콜의 실행이 먼저 완료되어야 하기 때문에 하위계층의 태스크는 상위 계층의 태스크들보다 높은 우선순위가 할당된다. 예를 들어, TCP 관련 태스크와 UDP 관련 태스크의 실행이 완료되기 위해서는 ARP 태스크와 IP 태스크의 실행이 먼저 완료되어야 한다.
   - 4-C. TCP 태스크는 전송 프로토콜 중에서 신뢰성 있는 통신을 담당하기 때문에

비신뢰 전송 프로토콜의 기능을 담당하는 UDP 태스크보다 높은 우선순위를 가진다.

4-D. 센서 노드의 경우 감지된 센싱 데이터의 전달이 빈번하게 실행되기 때문에 TCP 및 UDP 패킷의 수신보다는 TCP 및 UDP 패킷의 송신이 더 많이 발생한다. 이에 TCP-OUT 태스크와 UDP-OUT 태스크가 TCP-IN 태스크와 UDP-IN 태스크보다 높은 우선순위를 가진다.

표 3은 COSIP-I 모델 및 COSIP-II 모델, 그리고 COSIP-III 모델의 데이터 및 명령어 코드 크기를 보여준다. COSIP-I 모델은 멀티태스크기반 네트워크 컴포넌트의 실행을 위해 요구되는 추가적인 TCB(Task Control Block)와 스택에서 유지되는 태스크 정보의 저장 공간이 필요하지 않다. 그러나 COSIP-II 모델과 COSIP-III 모델에 서는 추가적인 태스크 정보의 유지 공간이 필요하게 된다. 특히, COSIP-III 모델은 비주기 태스크의 스케줄링을 위해 그림 2에서 제시한 모듈이 추가되었기 때문에 실행코드 크기순서는 COSIP-III 모델 > COSIP-II 모델 > COSIP-I 모델 순서로 된다. Atmel사의 Atmega128L 8비트 마이크로프로세서는 128K바이트의 명령어 메모리와 4K바이트의 데이터 메모리를 내장하고 있으며, 32K바이트의 데이터용 추가 외부 메모리 사용이 가능하다. 이에 27.25K바이트 명령어 코드와 11.52K바이트 데이터 코드를 가지는 COSIP-III 모델이 센서 노드 하드웨어 플랫폼에서 실행 가능함을 알 수 있다.

표 3. COSIP-I/II/III 모델간의 코드 사이즈 비교
(단위: K바이트)

테스트환경 / 코드유형	COSIP-I 모델	COSIP-II 모델	COSIP-III 모델
데이터	2.94	7.08	11.52
명령어	19.18	25.69	27.25
전체 실행 코드	22.13	32.77	38.77

그림 6은 COSIP-I 모델 및 COSIP-II 모델과 COSIP-III 모델에서 3개의 주기 응용 태스크(온도, 조도, 습도 측정 태스크)와 네트워크 태스크들이 동시에 실행될 때 측정된 네트워크 태스크들의 평균 응답시간을 보여 준다. COSIP-III 모델에서 수행된 네트워크 태스크들의 평균 응답시간은 COSIP-I 모델 및 COSIP-II 모델과 거의 유사하다. 그리고 COSIP-III 모델은 모든 주기 태스크의 마감시한 보장과 동시에 다중 TCP/UDP 통신 연결을 지원한다.

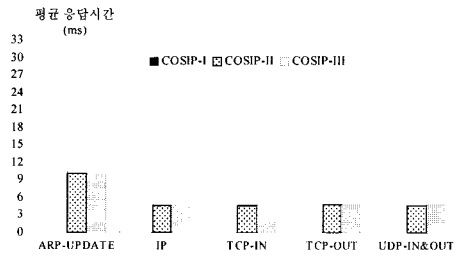

그림 6. 네트워크 태스크간의 평균 응답시간

그림 7은 비주기 BIG-AP 태스크를 포함한 모든 응용 태스크와 네트워크 태스크들이 동시에 실행될 때 측정된 평균 응답시간을 보여 준다. COSIP-III 모델에서 실행되는 주기 태스크인 온도, 조도, 습도 측정 태스크와 ARP-UPDATE 태스크는 COSIP-II 모델에서 실행되는 것보다 더 좋은 성능을 보여 주지만, COSIP-III 모델에서 실행되는 비주기 BIG-AP 태스크의 평균 응답시간은 COSIP-II 모델에서 실행되는 것보다 느리다. 이러한 현상이 발생하는 이유는 COSIP-III 모델에서는 비주기 BIG-AP 태스크의 간섭을 최소화하여 주기 태스크 마감시한을 보장한다. COSIP-II 모델에서 발생된 온도, 조도 측정 태스크의 마감시한 초과율은 각각 1.6%와 0.01%이었다.

그림 8과 그림 9에서는 보다 정밀한 실험 결과값을 보여주기 위하여 1ms당 100개의 클록 tick이 발생되는 동안에 측정된 값을 보여준다. 그림 8은 비주기 BIG-AP 태스크가 실행되지 않는 시험 환경에서 주기 태스크인 온도, 조도, 습도 측정 태스크가 COSIP-I/II/III 모델에서 수행된 후의 단위시간당 처리기 이용률을 보여 준다. COSIP-II 모델과 COSIP-III 모델에서는 네트워크 태스크간의 문맥교환이 발생하여 COSIP-I 모델보다 처리기 이용률이 높게 나올 수 있을 것이라고 추측할 수 있다. 그러나 동작 시험을 한 결과, COSIP-II 모델 및 COSIP-III 모델의 처리기 이용률은 COSIP-I 모델의 처리기 이용률과 거의 유사함을 확인하였다. 그림 9는 주기 태스크인 온도, 조도, 습도 측정 태스크와 비주기 BIG-AP 태스크가 동시에 실행되는 시

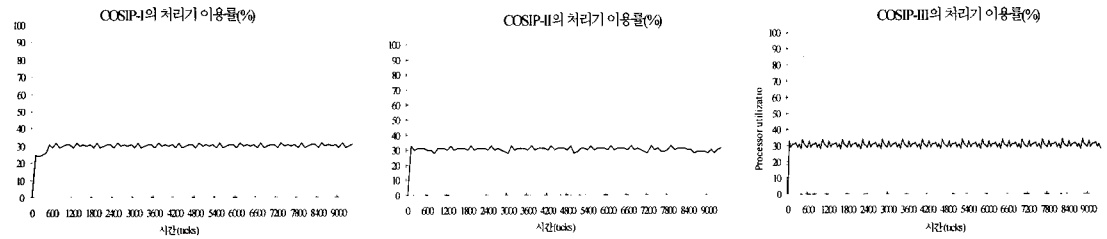

그림 8. 비주기 태스크 BIG-AP가 실행되지 않는 시험 환경에서 COSIP-I/II/III 모델의 단위시간당 처리기 이용률

험 환경에서 COSIP-II 모델과 COSIP-III 모델의 단위시간당 처리기 이용률을 보여준다. COSIP-III 모델에서는 비주기 태스크들이 처리기가 유휴상태인 경우에만 스케줄링이 가능하기 때문에 COSIP-III 모델의 처리기 이용률은 COSIP-II 모델보다 낮음을 확인하였다.

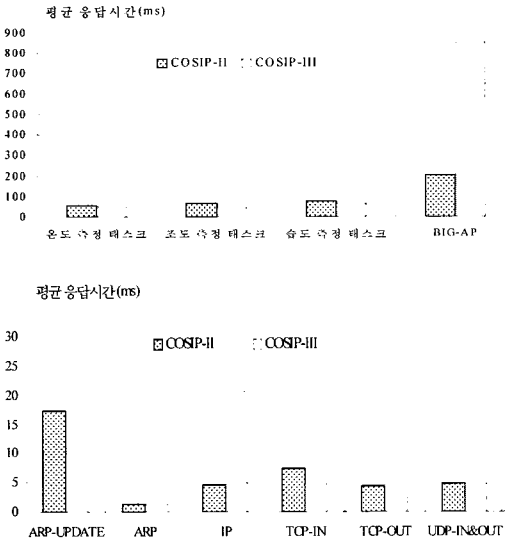

그림 7. 응용 태스크와 네트워크 태스크간의 평균 응답시간

그림 10은 COSIP-I 모델 및 COSIP-모델과 COSIP-III 모델의 평균 처리기 이용률을 보여 준다. 주기 태스크인 온도, 조도, 습도 측정 태스크와 비주기 BIG-AP 태스크가 동시에 실행될 경우에는 COSIP_III 모델의 평균 처리기 이용률이 COSIP-II 모델보다 더 좋은 성능을 제공할 수 있다. 이와 같이 동작 시험을 수행한 결과, COSIP-III 모델은 주기 태스크의 마감시한 보장뿐만 아니라 향상된 비주기 태스크의 평균 응답시간과 효율적인 평균 처리기 이용률을 제공할 수 있음을 확인하였다.

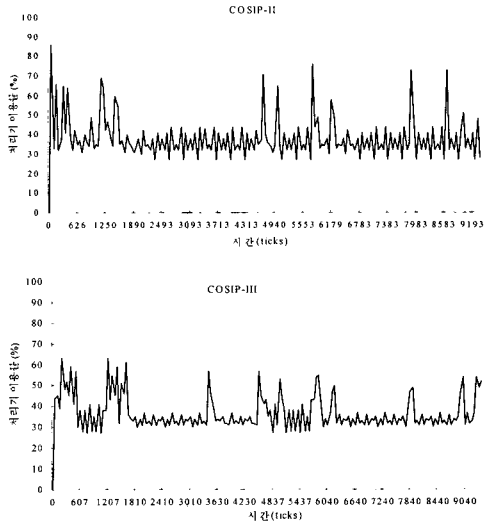

그림 9. BIG-AP가 실행되는 시험 환경에서 COSIP-II/III 모델의 단위시간당 처리기 이용률

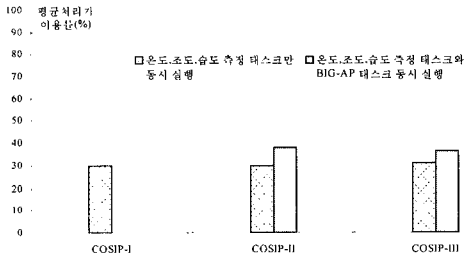

그림 10. COSIP-I/II/III 모델의 평균 처리기 이용률

V. 결 론

센서 노드 플랫폼을 위한 기존의 센서 네트워크 운영체제는 실시간성을 지원하지 못하거나 주기 태스크들의 스케줄링만 지원한다. 이와 같은 운영체제 환경에서는 주기 태스크의 마감시한 보장과 비주기 태스크의 빠른 응답시간이 요구되는 응용 서비스를 제공하기 위하여 개발자가 응용 계층에서

별도의 실시간 스케줄링 기법을 설계 및 구현해야 하는 문제점이 발생한다. 또한 기존의 센서 노드 플랫폼에서는 새로운 응용 및 네트워크 서비스의 추가와 삭제가 어렵기 때문에 빠른 개발 공정주기를 요구하는 센서 네트워크 산업에서 다양한 실시간 응용 서비스의 개발 및 관련 제품의 출시가 지연될 수 있다. 이에 본 논문에서는 멀티 태스크 기반의 확장성과 주기 및 비주기 태스크 관리 기법을 효율적으로 제공할 수 있는 실시간 센서 노드 플랫폼을 설계하고 구현하였다. 제안된 실시간 센서 노드 플랫폼을 상용 무선 센서 모트에서 동작 시험을 수행한 결과, 본 논문에서 제안한 실시간 태스크 관리 기법은 멀티 태스크 기반의 확장성과 주기 및 비주기 태스크의 실시간 관리 기법을 제공하기 위해 요구되는 부가적인 오버헤더가 있음에도 불구하고 주기 태스크의 마감시한 보장과 더불어 비주기 태스크의 빠른 응답시간과 효율적인 시스템의 평균 처리기 이용률을 제공할 수 있음을 확인하였다. 특히 운영체제 수준에서 주기 태스크의 마감시한 보장뿐만 아니라 비주기 태스크의 빠른 응답시간을 제공하기 때문에 센서 노드의 신뢰성을 향상시킬 수 있다. 이와 더불어 센서 노드의 용이한 확장성을 제공하기 위하여 멀티 태스크 기반의 동적 컴포넌트 실행 환경이 보장되는 센서 노드 플랫폼을 초경량 8비트 마이크로프로세서인 Atmel 사의 Atmega128L이 탑재된 센서 보드에서 설계 및 구현하여 제안된 기법에 대한 실용성을 검증하였다.

참 고 문 헌

[1] P. Levis, et al., "The Emergence of Networking Abstractions and Techniques in TinyOS," *Proc. of the First USENIX/ACM Symposium on Networked Systems Design and Implementation*, San Francisco, USA, pp. 1-14, March 2004.

[2] V. Subramonian, H. Huang-Ming, S. Datar, L. Chenyang, "Priority Scheduling in TinyOS - A Case Study," *Washington University Technical Report (WUCSE-2003-74)*, Washington University, USA, December 2003.

[3] S. Bhatti, et al., "Mantis OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms," *Mobile Networks and Applications(MONET) Journal*, vol. 10, no. 4, pp. 563-579, August 2005.

[4] T. J. Hofmeijer, S. O. Dullman, P. G. Jansen, P. J. Havinga, "DCOS, A Real-time Light-weight Data Centric Operating System," *Proc. of International Conference on Advances in Computer Science and Technology*, St. Thomas, USA, pp. 259-264, November 2004.

[5]. P. Ganesan, A.G. Dean, "Enhancing the AvrX kernel with efficient secure communication using software thread integration," *Proc. of the 10th Real-Time and Embedded Technology and Applications Symposium*, Toronto, Canada, pp. 265-275, May 2004.

[6] C. L. Liu, J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46-61, January 1973.

[7] K. Jeffay, C.U. Martel, "On non-preemptive scheduling of periodic and sporadic tasks," *Proc. of the 12th IEEE real-time systems symposium*, San Antonio, USA, pp. 129-139, December 1991.

[8] L. Georges, P. Muehlethaler, N. Rivierre, "A few results on non-preemptive real-time scheduling," *INRIA Research Report nRR3926*, May 2000.

[9] F. Cottet, J. Delacrioix, C. Kaiser, Z. Mammeri, *Scheduling in Real-time Systems*, John Wiley & Sons, December 2002.

[10] J. J. Labrosse, *MicroC/OS-II The Real-Time Kernel Second Edition*, CMP Books, April 2002.

[11] J. Lehoczky, L. Sha, Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," *Proc. of the 10th Real Time Systems Symposium*, California, USA, pp. 166-171, December 1989.

[12] J. P. Lehoczky, S. Ramos-Thuel, "An Optimal Algorithm for Scheduling Soft-aperiodic Tasks in Fixed-priority Preemptive Systems," *Proc. of the 13th Real-Time Systems Symp.*, pp. 110-123, December 1992.

[13] A. Dunkels, "Full TCP/IP for 8bit Architectures," *Proc. of the First ACM/Usenix International Conference on Mobile Systems*, San Francisco, USA, pp. 85-98, May 2003.

정 경 훈 (Kyung-hoon Jung) 정회원

1996년 2월 부경대학교 전자계산학과 졸업
1998년 2월 부경대학교 전자계산학과 석사
2006년 8월 부경대학교 전자계산학과 박사
2006년 9월~현재 부산대학교 U-Port정보기술산학공동사업단 전임연구원
<관심분야> 센서네트워크, 임베디드 시스템, 실시간 스케줄링, 분산병렬처리

김 병 훈 (Byoung-hoon Kim) 준회원

2006년 2월 부산대학교 정보컴퓨터공학부 졸업
2006년 3월~현재 부산대학교 컴퓨터공학과 석사과정
<관심분야> 메쉬 네트워킹, TCP/IP 통신, 실시간 시스템, 큐잉 이론

이 동 건 (Dong-geon Lee) 준회원

2005년 2월 신라대학교 컴퓨터교육과 졸업
2007년 2월 부산대학교 컴퓨터공학과 석사
2007년 3월~현재 부산대학교 컴퓨터공학과 박사과정
<관심분야> 무선 MAC, 실시간 스케줄링, QoS, SoC 설계

김 창 수 (Chang-soo Kim) 정회원

1984년 2월 울산대학교 전자계산학과 졸업
1986년 2월 중앙대학교 컴퓨터공학과 석사
1991년 2월 중앙대학교 컴퓨터공학과 박사
2002년~2003년 UMKC 방문 교수
1992년 3월~현재 부경대학교 전자컴퓨터정보통신공학부 교수
<관심분야> USN 방재시스템, LBS/GIS, Semantic GIS 검색, 임베디드 시스템

탁 성 우 (Sung-woo Tak) 종신회원

1995년 2월 부산대학교 컴퓨터공학과 졸업
1997년 2월 부산대학교 컴퓨터공학과 석사
2003년 2월 미국미주리주립대학교 Computer Science박사
2004년~현재 부산대학교 정보컴퓨터공학부 조교수
2004년~현재 부산대학교 컴퓨터 및 정보통신 연구소 겸임 연구원
<관심분야> 유무선 네트워크, SoC 설계, 실시간 시스템, 위치인식, 최적화 기법, 그래프 이론, 큐잉 이론