유비쿼터스 응용 개발을 위한 센서 네트워크 시뮬레이터

(Sensor Network Simulator for Ubiquitous Application Development)

김 방 현 † 김 종 현 ††
(Bang-Hyun Kim) (Jong-Hyun Kim)

요 약 유비쿼터스 컴퓨팅의 인프라가 되는 무선 센서 네트워크의 설계 및 응용 개발을 위하여 소프트웨어 시뮬레이션이 널리 사용되고 있다. 본 연구에서는 센서 네트워크 응용프로그램의 동작을 확인할 수 있고, 실행시간 및 전력소모량을 예측할 수 있으며, 많은 수의 센서노드들을 시뮬레이션 할 수 있는 센서 네트워크 시뮬레이터를 개발하였다. 시뮬레이터는 명령어 수준의 병렬 이산 사건 시뮬레이션 방법을 이용하여 구현되었다. 명령어 수준의 시뮬레이션은 실제 센서보드에 적재되는 실행이미지를 시뮬레이션 작업부하로 사용하기 때문에 시뮬레이션 정밀도가 높다. 병렬 시뮬레이션은 여러 대의 컴퓨터를 사용하여 작업부하를 분산 처리하므로 대규모의 센서 네트워크를 시뮬레이션 할 수 있게 해준다. 구현된 시뮬레이터는 센서보드 내의 모듈 별 동작시간 및 실행된 명령어 수를 근거로 하여 전력소모량을 예측할 수 있다. 또한 다양한 시나리오의 유비쿼터스 응용프로그램의 수행 과정을 시뮬레이션 할 수 있으며, 디버깅도 가능하다. 이 연구에서 시뮬레이션의 작업부하인 명령어 트레이스로는 ATmega128L 마이크로컨트롤러용 크로스컴파일러에 의해 생성된 실행이미지를 사용하였다.

키워드 : 센서 네트워크 시뮬레이터, 명령어 시뮬레이션, 병렬 시뮬레이션

Abstract Software simulations have been widely used for the design and application development of a wireless sensor network that is an infrastructure of ubiquitous computing. In this study, we develop a sensor network simulator that can verify the behavior of sensor network applications, estimate execution time and power consumption, and simulate a large-scale sensor network. To implement the simulator, we use an instruction-level parallel discrete-event simulation method. Instruction-level simulation uses executable images loaded into a real sensor board as workload, such that it results in the high degree of details. Parallel simulation makes simulation of a large-scale sensor network possible by distributing workload into multiple computers. The simulator can predict the amount of power consumption based on operating time of modules in a sensor node and counting the number of executed instructions by kind. Also it can simulate ubiquitous applications with various scenarios and debug programs. Instruction traces used as workload for simulations are executable images produced by the cross-compiler for ATmega128L microcontroller.

Key words : sensor network simulator, instruction-level simulation, parallel simulation

† 학생회원 : 연세대학교 컴퓨터정보통신공학부
legnamai@chol.com
†† 종신회원 : 연세대학교 컴퓨터정보통신공학부 교수
jhkim34@yonsei.ac.kr
논문접수 : 2007년 9월 6일
심사완료 : 2007년 10월 26일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제13권 제6호(2007.11)

1. 서 론

유비쿼터스 컴퓨팅의 인프라가 되는 무선 센서 네트워크(wireless sensor networks: 이하 WSN이라 함)는 제한된 전력량을 가진 매우 작은 하드웨어로 이루어지는 많은 수의 센서노드들로 구성된다. 이 네트워크의 토폴로지와 라우팅 방식은 그 목적에 따라 결정되어야 하며, 하드웨어 및 소프트웨어도 필요한 경우에는 변경되어야 한다. 시뮬레이션은 시스템이 구현되기 전에 시스템 동작과 성능을 예측할 수 있게 한다. 따라서 WSN을

최적으로 설계하기 위해서는 WSN 시뮬레이터가 필요하다[1].

그러한 목적을 위해 WSN 시뮬레이터는 다음과 같은 세 가지 기능들을 제공해야 한다. 첫째, 센서노드에 적재되는 응용프로그램의 실행을 가능한 한 정확하게 시뮬레이션 함으로써, WSN의 동작을 확인할 수 있어야 한다. 둘째, 센서노드들이 가진 전력량이 한정되어 있기 때문에, 응용프로그램의 실행시간 및 전력소모량을 예측할 수 있어야 한다. 마지막으로, WSN은 많은 수의 센서노드들을 시뮬레이션 할 수 있어야 한다. 그러나 WSN 시뮬레이션에 대한 지금까지의 대부분 연구들은 WSN의 동작을 확인하는 것에만 집중되어 있으며, 특정한 WSN 운영체제나 라이브러리 API, 또는 프로그래밍 언어가 필요하다는 문제점이 있다. 예를 들어, 널리 알려진 WSN 시뮬레이터인 TOSSIM[2]의 경우에는 WSN 운영체제인 TinyOS[3] 기반의 응용프로그램들에 대해서만 시뮬레이션이 가능하며, 실행시간과 전력소모량은 예측할 수 없다. 그리고 Avrora[4]의 경우에는 실행시간과 전력소모량은 예측할 수 있지만, 센서노드를 하나의 쓰레드로 동작시키기 때문에 많은 수의 센서노드들을 시뮬레이션 하기가 어렵다.

본 연구에서는 위의 세 가지 기능들을 모두 충족시킬 수 있는 명령어 수준의 이산 사건 시뮬레이션(instruction-level discrete-event simulation) 방법을 제안한다. 명령어 시뮬레이션은 센서노드의 시스템 클록(clock) 수준으로 시뮬레이션의 정밀도를 높임으로써, WSN 응용프로그램의 실행동작과 실행시간 및 전력소모량을 실제 센서보드와 거의 근접한 수준으로 예측할 수 있다. 또한 이 방법은 실제 센서노드에 적재되는 실행이미지(executable image)를 시뮬레이션 작업부하로 사용하여 기존의 많은 WSN 시뮬레이터들이 가진 소프트웨어 의존성 문제를 해결하였다. 명령어 시뮬레이션은 시뮬레이션 정밀도가 높기 때문에 시뮬레이션 시간이 오래 걸리지만, 최적 동기식(optimal-synchronous)의 병렬 이산 사건 시뮬레이션 방법(parallel discrete-event simulation: 이하 PDES라 함)을 사용하여 시뮬레이션 시간을 단축하였다.

본 연구에서 제안한 명령어 수준의 이산 사건 시뮬레이션 방법과 최적 동기식 PDES 방법을 사용하여 개발한 병렬 센서 네트워크 시뮬레이터(parallel sensor network simulator: 이하 PASENS라 함)는 이전 연구에서 구현한 MISS(machine instruction-level sensor network simulator)[1]의 문제점이었던 느린 시뮬레이션 속도를 병렬 시뮬레이션을 이용하여 개선한 시뮬레이터이다. 따라서 PASENS는 높은 시뮬레이션 정밀도를 유지하는 동시에 시뮬레이션 시간을 크게 단축시켜서 많은 수의 센서노드들을 포함하는 대규모 센서 네트워크를 단시간 내에 시뮬레이션 할 수 있다. 특히 PASENS에는 사용자가 유비쿼터스 응용을 효율적으로 개발할 수 있도록 디버깅 기능과 다양한 시나리오를 시뮬레이션 할 수 있는 기능, 그리고 실행된 명령어들을 종류별로 계수할 수 있는 기능과 같은 사용자를 위한 많은 기능들이 추가되었다.

본 논문의 구성은 2장에서 기존의 WSN 시뮬레이션 방법들을 간략히 소개하고, 3장에서는 명령어 수준의 이산 사건 시뮬레이션 방법을 설명하였다. 4장에서는 제안한 방법으로 구현한 시뮬레이터를 사용하여 실험한 결과들을 제시하고 분석하였다. 마지막으로 5장에서는 전반적인 결론을 요약하였다.

2. 관련 연구

그동안 많은 WSN 연구들에서는 ns-2[5]와 같은 기존의 범용 네트워크 시뮬레이터를 이용하여 프로토콜이나 알고리즘을 평가해 왔다. 이러한 예로는 ns-2에 사용된 802.11 MAC 프로토콜로 센서 네트워크를 시뮬레이션 한 연구[6]와 50~200개의 노드에서 802.11 MAC 프로토콜 시뮬레이션을 사용한 지리적 해쉬 테이블(geographic hash table)에 관한 연구[7]가 있다. Monarch(mobile networking architectures) 프로젝트[8]는 무선이동통신에 적합하도록 ns-2를 확장하였고, SensorSim[9]은 실제 센서보드의 전력 및 통신 프로토콜 시뮬레이션을 위하여 ns-2의 802.11 네트워크 모델을 확장하였다. 이 시뮬레이터들은 네트워크의 프로토콜 및 알고리즘은 시뮬레이션 할 수 있지만, 실제 센서보드에 적재되는 응용프로그램의 동작은 시뮬레이션 할 수 없다.

WSN 시뮬레이터에 대한 많은 연구가 NEST(network embedded software technology) 프로젝트[10]에서 이루어졌는데, 그 결과로서 TOSSIM(TinyOS simulator)[2], Prowler(probabilistic wireless network simulator)[11], Siesta(simple NEST application simulator)[12], 그리고 Ashut(acoustic simulator for urban terrain)[10]와 Rmase(routing modeling application simulation environment)[10]가 출현하였다. 이들은 모두 NEST 프로젝트에서 개발된 TinyOS[3]가 탑재되어 있는 하드웨어 플랫폼 기반에서 동작하는 시뮬레이터들이다. TOSSIM은 센서노드에 탑재된 TinyOS와 응용프로그램의 동작 및 상호작용을 시뮬레이션 하는 운영체제 시뮬레이터이고, Prowler는 무선 네트워크 알고리즘을 평가할 수 있는 네트워크 시뮬레이터이며, Siesta는 NEST 응용 프로그램 및 미들웨어의 기능을 검증할 수 있는 미들웨어 시뮬레이터이다. 그리고 Ashut는 주어진 환경에서 직접 전달되는 음향과 반사 전달되는 음향의 도착 시간을 계산할 수 있는 시뮬레이터이고, Rmase는

네트워크 토폴로지 및 응용 시나리오를 생성하여 센서 네트워크의 라우팅 기능을 평가하는 Prowler 기반의 라우팅 시뮬레이터이다. 그러나 이 시뮬레이터들은 TinyOS 기반에서만 동작하며, 실행시간 및 전력소모량을 예측할 수 없다. 특히 TOSSIM을 구동하기 위해서는 TinyOS에서 TOSSIM용으로 컴파일된 작업부하가 필요하고, Prowler의 경우에는 MATLAB 소프트웨어가 필요하며, Siesta는 시뮬레이션을 하기 위해서는 응용 소프트웨어가 특정한 라이브러리 API를 포함해야 한다.

TOSSIM을 기반으로 하여 시뮬레이션 할 수 있는 센서노드 수의 확장성을 높인 TOSSF(TinyOS scalable simulation framework)[13]는 TinyOS 코드를 병렬 시뮬레이션이 가능한 SWAN(simulator for wireless ad-hoc network)[14] 프레임워크로 컴파일하여 대규모 WSN을 시뮬레이션 할 수 있다. 특히 TOSSF는 TOSSIM과는 달리 센서노드에 서로 다른 실행이미지를 적재할 수 있고 동적 네트워크 토폴로지를 구성할 수 있다. TOSSF와 유사한 형태인 PowerTOSSIM[15]은 TinyOS의 컴포넌트 단위로 실측한 전력소모량을 사용하여 평균 오차율 4.7%의 비교적 정확한 전력소모량을 예측할 수 있으며, Tython[16]은 TOSSIM에 Python [17] 기반의 스크립트를 결합하여 다양한 응용 시나리오를 재생산할 수 있는 동적 시뮬레이션 환경을 제공한다. 이 시뮬레이터들은 부분적으로 TOSSIM의 문제점을 개선하였지만, 여전히 TOSSIM용으로 컴파일된 작업부하만을 사용해야 하는 문제점을 가지고 있다.

SENS(a sensor, environment and network simulator)[18]는 응용프로그램, 네트워크 통신, 그리고 물리적 환경의 모듈로 구성된 WSN 응용에 대하여 유연성 있는 시뮬레이션 환경을 제공하며, TOSSF와 마찬가지로 TinyOS의 라이브러리 API를 SENS의 라이브러리 API로 연결하여 사용할 수 있다. 한편 EmStar[19]는 HP사의 iPAQ 혹은 Crossbow사의 Stargate와 같은 마이크로서버에서 실행되는 WSN 응용을 개발할 수 있도록 소프트웨어 환경을 제공한다. EmStar에 포함된 도구인 EmSim과 EmCee는 WSN 시뮬레이션 환경을 제공한다. 그러나 이들 시뮬레이터들은 각 시뮬레이터가 제공하는 특정한 API를 사용해야만 시뮬레이션이 가능하다.

ATEMU[20]와 Avrora[4]는 시뮬레이션 정밀도가 높은 명령어 수준의 시뮬레이터로서 본 연구에서 제안한 명령어 수준의 이산 사건 시뮬레이션 방법과 유사한 방법을 사용하는 시뮬레이터들이다. 본 연구의 명령어 시뮬레이션이 이산 사건 시뮬레이션 방법을 사용하는 반면에, ATEMU는 센서노드의 시스템 클록 간격으로 가상 센서노드들에 적재된 실행이미지를 순차적으로 시뮬레이션 하고, Avrora는 각 센서노드가 하나의 쓰레드(thread)로 동작하는 다중 쓰레드 방식으로 시뮬레이션 한다. 이 시뮬레이터들은 실행시간 및 전력소모량은 예측할 수 있지만, 높은 시뮬레이션 정밀도로 인하여 길어진 시뮬레이션 시간 때문에 많은 수의 센서노드들을 시뮬레이션 하기가 어렵다.

3. 명령어 수준의 이산 사건 시뮬레이션

3.1 개요

WSN 시뮬레이션에서 명령어 시뮬레이션 방법은 실제 센서노드에 적재되는 실행이미지를 작업부하로 사용하며, 센서노드에서 하드웨어적으로 발생하는 동작들을 시스템 클록 수준으로 기계명령어들을 시뮬레이션 한다. 이러한 특성으로 인하여 명령어 시뮬레이션 방법은 WSN 응용프로그램들의 동작과 실행시간 및 전력소모량을 실제 센서보드와 거의 근접한 수준으로 예측할 수 있다. 또한 이 방법은 WSN 운영체제나 라이브러리 API, 또는 프로그래밍 언어에 상관없이 실제 센서노드에 적재되는 실행이미지만 있으면 시뮬레이션이 가능하다. 즉, 명령어 시뮬레이션 방법은 기존의 많은 WSN 시뮬레이터들이 가진 소프트웨어 의존성 문제를 해결할 수 있다. 비록 하드웨어 의존성 문제가 발생하지만, 일반적으로 WSN에서 센서노드의 하드웨어 구조들은 소프트웨어에 비하여 상대적으로 종류가 적기 때문에 명령어 시뮬레이션은 의존성 측면에서 얻을 수 있는 장점이 단점보다 더 큰 방법이다.

명령어 시뮬레이션을 사용한 WSN 시뮬레이터들로는 ATEMU[20]와 Avrora[4], 그리고 본 연구의 PASENS가 있다. PASENS가 이산 사건 시뮬레이션 방법을 사용하여 명령어 시뮬레이션을 하는 반면에, ATEMU는 센서노드의 시스템 클록 간격으로 가상 센서노드들에 적재된 실행이미지를 순차적으로 시뮬레이션 하고, Avrora는 각 센서노드가 하나의 쓰레드로 동작하는 다중 쓰레드 방식으로 시뮬레이션 한다. 그리고 PASENS는 NEST 프로젝트[10]에서 개발된 MICAz 형식의 센서보드를 시뮬레이션 할 수 있는 반면에, ATEMU와 Avrora는 같은 프로젝트에서 MICAz 이전에 개발되었던 Mica2 형식의 센서보드를 시뮬레이션 할 수 있다. MICAz 형식의 센서보드는 RF 트랜시버로 Chipcon사의 CC2420을 사용하는 점을 제외하면, 같은 회사의 CC1000을 사용하는 Mica2 형식의 센서보드와 동일한 하드웨어 구조를 가지고 있다.

명령어 시뮬레이션은 시뮬레이션 정밀도가 높기 때문에 시뮬레이션 시간이 오래 걸리며, 시뮬레이션 되는 센서노드의 수가 많아지면 그 시간은 더 길어지는 문제점이 있다. Avrora는 시뮬레이션 시간을 단축시키기 위하여 슬립(sleep) 상태와 타이머 등을 시뮬레이션 할 때

사건 큐(event queue)를 사용하고, 쓰레드들을 관리하기 위한 효율적인 동기화 방법을 사용하였다. 또한 Avrora는 다중 쓰레드를 다중 프로세서 상에서 동작시킴으로써 시뮬레이션 시간을 단축하려 하였지만, 프로세서 수가 증가하면 전역 데이타 구조에 대한 경합(contention)과 운영체제의 쓰레드 스위칭 오버헤드 때문에 제한적인 속도향상만을 얻을 수 있었다. Avrora[4]의 실험결과에 따르면 ATEMU는 8시간 안에 512개의 센서노드를 시뮬레이션 하지 못하였고, Avrora는 같은 수의 센서노드들을 5분 안에 시뮬레이션 하였다. 그러나 Avrora는 네 개의 센서노드를 네 개의 프로세서에서 시뮬레이션 한 경우에 2배의 속도향상을 얻은 반면에, 32개의 센서노드를 8개의 프로세서에서 시뮬레이션 한 경우에는 속도향상이 1.8배였다. 이러한 결과는 쓰레드 기반의 시뮬레이션은 병렬 시뮬레이션에 부적합하다는 것을 나타낸다.

본 연구에서는 명령어 시뮬레이션을 위하여 이산 사건 시뮬레이션 방법을 사용한다. 이산 사건 시뮬레이션은 시뮬레이션 대상의 동작을 사건 단위로 정의하고, 그 동작이 발생할 시간과 함께 사건을 사건 큐에 삽입한다. 시뮬레이션은 사건 큐에 있는 사건들 중에서 가장 빨리 발생되는 사건을 사건 큐에서 꺼내면서 진행되기 때문에, 이산 사건 시뮬레이션에서는 별도의 동기화 방법이 필요가 없다. 즉, 센서노드 내부의 모듈들의 동작들에 대하여 동작의 발생 시간과 상호 작용에 따라 사건을 정의하면, 이산 사건 시뮬레이션은 이들 동작들이 실제 센서보드에서 발생되는 시간 순서와 동일하게 시뮬레이션을 수행한다. 또한 센서노드들 사이의 동작들도 이와 동일한 이유로 인하여 동기화가 이루어진다.

명령어 수준 시뮬레이션에서 이산 사건 시뮬레이션 방법을 사용할 경우에는, 병렬 이산 사건 시뮬레이션 방법들을 적용하여 병렬 시뮬레이션이 가능하다. 병렬 시뮬레이션은 명령어 수준 시뮬레이션의 문제점인 느린 시뮬레이션 속도를 향상시킴으로써, 많은 수의 센서노드를 갖고 있는 WSN을 시뮬레이션 할 수 있게 한다.

3.2 시뮬레이터의 구조

본 연구에서 개발한 시뮬레이터인 PASENS는 그림 1에서 보는 바와 같이 쉘 에이전트(shell agent), 통신 에이전트(communication agent), 단일 시뮬레이션 제어기(single simulation controller), 병렬 시뮬레이션 제어기(parallel simulation controller), 그리고 가상 센서노드(virtual sensor node)로 구성되어 있으며, C 언어로 구현되었다. 쉘 에이전트는 PASENS를 쉘 명령어 기반으로 제어하는 모듈이다. 쉘 명령어는 키보드를 통해 사용자로부터 입력되거나, 통신 에이전트를 통하여 같은 컴퓨터 혹은 다른 컴퓨터의 프로그램으로부터 입력된다.

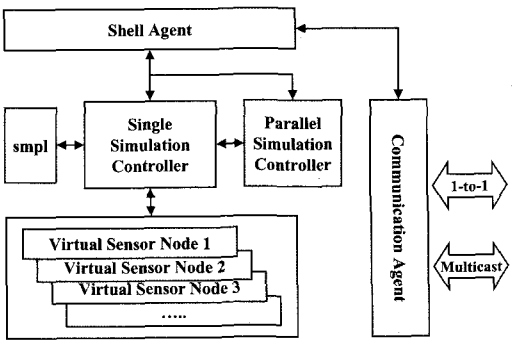


그림 1 PASENS의 구조

쉘 에이전트는 입력된 쉘 명령어를 해석하고, PASENS의 다른 모듈들을 제어하며 해석된 명령어를 실행한다. 또한 쉘 에이전트는 다른 컴퓨터에서 실행되는 PASENS 프로그램에게 쉘 명령어를 전달하여 원격 동작을 수행할 수도 있다. 병렬 시뮬레이션에서는 이러한 방법을 통하여 쉘 에이전트가 다른 컴퓨터에서 실행되는 PASENS의 에이전트와 연동한다. 통신 에이전트는 PASENS 프로그램들 사이의 통신과 PASENS를 이용하는 다른 응용프로그램들 사이의 통신을 위하여 TCP 기반의 일대일통신(1-to-1) 혹은 UDP 기반의 멀티캐스트(multicast) 통신을 수행하는 모듈이다.

단일 시뮬레이션 제어기는 사건 큐에서 사건을 인출하여 처리하고, 그 사건에 의해 스케줄 된 새로운 사건을 사건 큐에 삽입함으로써 가상 센서노드에서 명령어 수준의 이산 사건 시뮬레이션을 수행하는 모듈이다. 대부분의 사건 루틴들은 가상시간(virtual time, simulated time: 이하 VT라 함)을 증가시키는 동작들로 구성된다. 예를 들어, 사건 큐에서 꺼내진 "명령어 인출" 사건은 해당 명령어의 실행시간 후에 그 명령어가 실행되도록 "명령어 실행" 사건을 스케줄 한다. 그리고 VT가 "명령어 실행" 사건의 발생 시간에 도달하면 "명령어 실행" 사건이 사건 큐에서 꺼내져 실행된다. 사건 큐의 제어는 이산 사건 시뮬레이션을 위한 라이브러리인 smpl[21]을 이용하였다. 병렬 시뮬레이션 제어기는 여러 개의 컴퓨터들에서 병렬 시뮬레이션을 진행할 때에 준비 작업과 동기화 작업을 수행한다.

가상 센서노드는 센서노드의 동작을 소프트웨어적으로 시뮬레이션 하기 위한 모듈로서, NEST 프로젝트[10]에서 개발된 MICAz 형식의 WSN 센서보드인 CrossBow 사의 MPR2400[22] 보드와 Octacomm 사의 NANO-24[23] 보드를 모델링 하였다. MICAz 형식의 센서보드에는 Atmel사의 ATmega128L[24] 마이크로컨트롤러(microcontroller: 이하 MCU라 함)와 Chipcon사의 CC2420[25] RF 트랜시버가 장착되어 있다.

ATmega128L은 MCU의 클록이 16 ㎒에서 최대 16 MIPS의 처리율(throughput)을 가진 AVR Enhanced RISC(reduced instruction set computer) 구조의 8 비트 MCU이고, CC2420은 250 Kbps로 데이타를 전송할 수 있는 2.4 GHz IEEE 802.15.4 호환 RF 트랜시버이다. 센서보드를 가상 하드웨어 단위로 설정한 이유는 WSN에서는 센서보드가 하드웨어적으로 독립된 개체이기 때문이다. 대부분의 센서보드들은 MCU와 RF 트랜시버를 기본 요소로 하여 구성되기 때문에, 유사한 형태의 센서보드들은 약간의 수정만을 통하여 시뮬레이션이 가능하다. 예를 들어, 본 연구에서 가상 센서노드의 모델로 사용한 MPR2400 보드와 NANO-24 보드의 차이점은 센서보드 내의 모듈들을 연결하는 Atmega128L의 외부 핀 배정만 차이가 있을 뿐이다.

ATmega128L의 기억장치는 128 Kbyte 플래시 기억장치, 4 Kbyte EEPROM, 32개의 범용 레지스터(general purpose register), 64개의 I/O 레지스터, 160개의 확장 I/O 레지스터, 그리고 4 Kbyte의 내부 SRAM이 내장되어 있으며, 최대 64 Kbyte의 외부 SRAM을 연결할 수 있다. 그리고 ATmega128L의 주변장치로는 2개의 8 비트 타이머, 2개의 16 비트 타이머, SPI(serial peripheral interface), 2개의 USART(universal synchronous and asynchronous serial receiver and transmitter), TWI(two-wire serial interface), 그리고 8개의 채널과 10 비트의 분해능(resolution)을 갖는 ADC(analog to digital convertor)가 있다[24].

CC2420의 기억장치는 33개의 16 비트 레지스터와 368 Byte의 RAM이 내장되어 있고, 사용자는 15개의 스트로브 명령 레지스터(strobe command register)를 이용하여 CC2420을 제어할 수 있다[25]. 그 외에 MICAz 형식의 센서보드에는 적색, 녹색, 그리고 황색의 LED가 있고 온도센서, 조도센서, 습도센서, 그리고 초음파센서 등과 같은 센서들을 연결할 수 있는 인터페이스가 있다. 그림 2는 이와 같은 MICAz 형식의 센서노드를 명령어 시뮬레이션에 적합하도록 구성한 가상 센서노드의 구조이다. 가상 센서노드는 하드웨어의 기억장치를 소프트웨어로 추상화한 모듈이며, 하나의 가상 센서노드는 138KB 크기의 동적 기억장치(dynamic memory)를 사용하여 구현되었다.

3.3 명령어 수준 시뮬레이션

PASENS가 사용하는 시뮬레이션 작업부하는 실제 센서보드에 적재되는 실행이미지로서, ATmega128L MCU용으로 크로스 컴파일되어 센서보드에 적재되는 인텔 헥스 레코드 형식(Intel's hex-record format: .hex), 모토롤라 S 레코드 형식(S-record format: .srec), 그리고 롬 이미지 형식(ROM image format: .rom)의 파일을 사용한다. 명령어 수준 시뮬레이션 방법은 이 실행 이미지를 시뮬레이션 엔진의 가상 센서노드로 적재한 다음에, 실제 센서노드에서 프로세서 명령어가 실행되는 방법과 거의 동일하게 이루어진다. 즉, 명령어 실행동작의 시뮬레이션은 실제 센서노드에서와 같이 가상 플래시 메모리에서 현재 프로그램 카운터(program counter, PC)가 가리키는 위치의 기계명령어를 인출하고, 기계명령어의 비트 패턴을 분석하여 실행함으로써 이루어진다. 만약 명령어가 실행되는 중에 인터럽트가 발생되면, 인터럽트를 처리하기 위하여 명령어의 실행이 종료되는 VT에 수행되는 사건이 생성되어 사건 큐에 삽입된다. 그림 3은 PASENS가 센서노드의 MCU에서 실행되는 명령어의 동작을 가상 센서노드에서 명령어 수준으로 시뮬레이션 하는 개념도를 보여주고 있다.

그림 4는 두 개의 기계명령어 비트 패턴을 분석하고 실행하는 명령어 수준 시뮬레이션의 예를 보여주고 있다. "0000 1101 1010 1011"의 비트 패턴을 가지는 첫 번째 기계명령어는 상위 6 비트 "1100 11"이 연산코드(opcode) 규칙에 따라 산술명령어 "ADD" 명령어로 해석되고, "ADD" 명령어의 오퍼랜드(operand) 규칙에 따라 오퍼랜드는 "R28, R11"로 해석된다. 따라서 시뮬레이션 동작은 가상 센서노드의 레지스터 R28의 값에

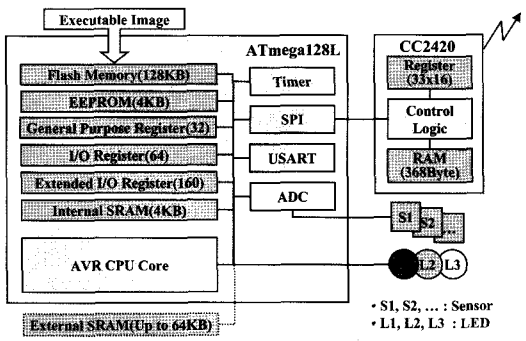


그림 2 MICAz 형식의 가상 센서노드

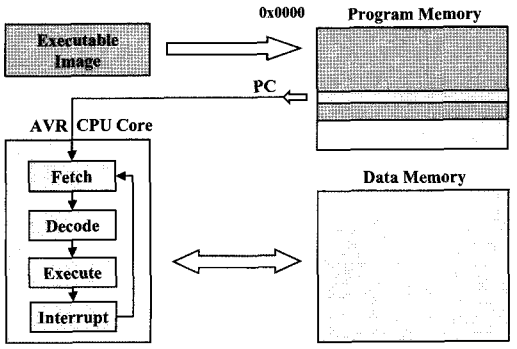
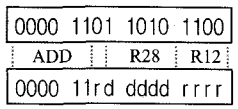
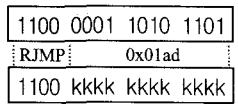


그림 3 명령어 시뮬레이션의 개념도



[해석된 명령] **ADD R28, R12**

[수행된 연산] **node->r[28] = node->r[28] + node->r[12];**



[해석된 명령] **RJMP 0x01ad**

[수행된 연산] **node->pc = node->pc + 0x01ad + 1;**

그림 4 명령어 시뮬레이션의 예

R11의 값을 더하고, 그 결과값을 R28에 저장하는 것으로 이루어지며, VT는 1 사이클이 증가한다. 또한 그림 4에는 나타나 있지 않지만, 산술명령어의 경우에는 연산결과에 따라 상태 레지스터(status register)의 내용이 변경된다. "1100 0001 1010 1100"의 비트 패턴을 가진 두 번째 기계명령어는 상위 4 비트 "1100"이 연산코드 규칙에 따라 산술명령어 "RJMP" 명령어로 해석되고, "RJMP" 명령어의 오퍼랜드 규칙에 따라 오퍼랜드는 "0x01ad"로 해석된다. 따라서 시뮬레이션의 실행은 가상 센서노드의 PC의 값을 "PC + 0x01ad + 1"로 변경하는 것으로 이루어지며, VT는 그 명령어의 실제 실행시간에 대응되는 2 사이클이 증가한다. 이러한 MCU 사이클 단위의 명령어 수준 시뮬레이션을 이용하면 비교적 정확한 실행시간을 예측할 수 있으며, 실행시간을 기반으로 하여 전력소모량을 예측할 수 있다.

3.4 이산 사건 시뮬레이션

이산 사건 시뮬레이션은 VT의 진행이 사건의 발생 및 그 소요 시간에 의해서 이루어지게 하는 방법이다. 즉, 사건이 발생하면 그 과정에 걸리는 시간만큼 지난 시간으로 VT가 진행되는 것이다. 그리고 이산 사건 시스템의 상태(state)는 사건 발생에 의해 불규칙한 시간 간격으로 전이되며, 구체적인 물리적 상황을 표현하는 상징적 의미를 가진다. 다시 말하면, 이산 사건 시뮬레이션은 모델의 상태 변화가 일어나는 경우에만 시간을 진행시키는 방법이다[26].

순차적으로 진행되는 시뮬레이션이 객체중심 시스템 내에서의 물리적인 진행에 초점이 맞춰진 반면, 이산 사건 시뮬레이션은 논리적인 시스템의 상태와 사건의 발생으로 인한 상태의 변화에 따라서 논리적으로 진행된다는 것에 큰 차이가 있다. 이산 사건 시뮬레이션에서의 사건은 상태의 주체가 되는 객체에게 상태적인 변화를 일으키도록 하는 원인이 된다. 이 상태는 입력 사건에

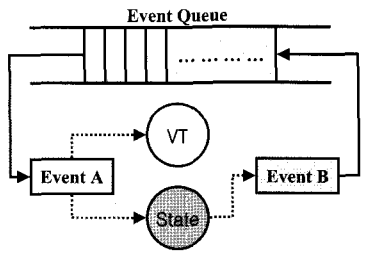


그림 5 이산 사건 시뮬레이션의 기본 개념

의하여 변화의 결과인 출력 사건을 내보내며, 이러한 사건들이 스케줄링 되어 사건 리스트(event list)를 형성한다. 상태는 사건으로 인하여 어느 이산 지점에서 변화를 일으킨다. 이로 인하여 상태의 변화가 발생하면 그에 따른 시간의 경과가 시간 스탬프(time stamp)에 표시된다. 결국 전체의 시뮬레이션은 이 리스트에 의해서 계획되며 진행되는 것이다. 그림 5는 그와 같은 이산 사건 시뮬레이션의 기본 개념을 보여주고 있다.

본 연구에서는 가상 센서노드의 내부 모듈 및 가상 센서노드들 사이의 시간 동기화를 위하여 이산 사건 시뮬레이션 방식을 이용한다. 특히 본 연구의 대상인 WSN은 규모가 커지는 경우에는 센서노드의 수가 수천 개에 이를 수도 있기 때문에 이러한 시간 동기화는 매우 중요하다. 또한 대부분의 WSN 응용프로그램은 전력소모를 최소화하기 위하여 동작 시간의 95% 이상이 슬립 상태가 되도록 구현되기 때문에, 명령어가 실행되지 않는 슬립 상태에서는 사건이 발생하지 않는 이산 사건 시뮬레이션 방식을 사용하면 시뮬레이션의 효율을 높일 수 있다.

그림 6은 ATmega128L MCU에 의한 명령어 실행, 인터럽트 처리, 타이머 동작, EEPROM 쓰기, SPI 동작, 그리고 SPI 동작에 따른 CC2420 RF 트랜시버의 RF 통신 등을 시뮬레이션 하기 위한 사건 루틴들 사이의 스케줄링 관계와 시간을 보여주고 있다. 대부분의 사건들은 VT를 증가시키는 동작들로 정의되며, 이 사건 루틴들은 동작의 순차성에 따라 적절한 사건 시간(event time) 간격으로 관련 사건을 스케줄링 하게 된다. 스케줄링 된 사건들은 시간 스탬프와 함께 사건 큐에 저장된다. 그리고 그들 중에서 가장 빠른 시간 내에 발생해야 하는 즉, 시간 스탬프 값이 가장 적은 사건부터 제거되어 처리되며, VT는 그 시간으로 갱신된다. 주요 몇 가지 사건 루틴들에 대하여 좀 더 상세히 설명하면 다음과 같다.

- INSTRUCTION_START 루틴

MCU가 명령어를 인출하고 실행을 준비하는 루틴으로서, 가상 센서노드의 플래시 메모리에 적재되어 있는

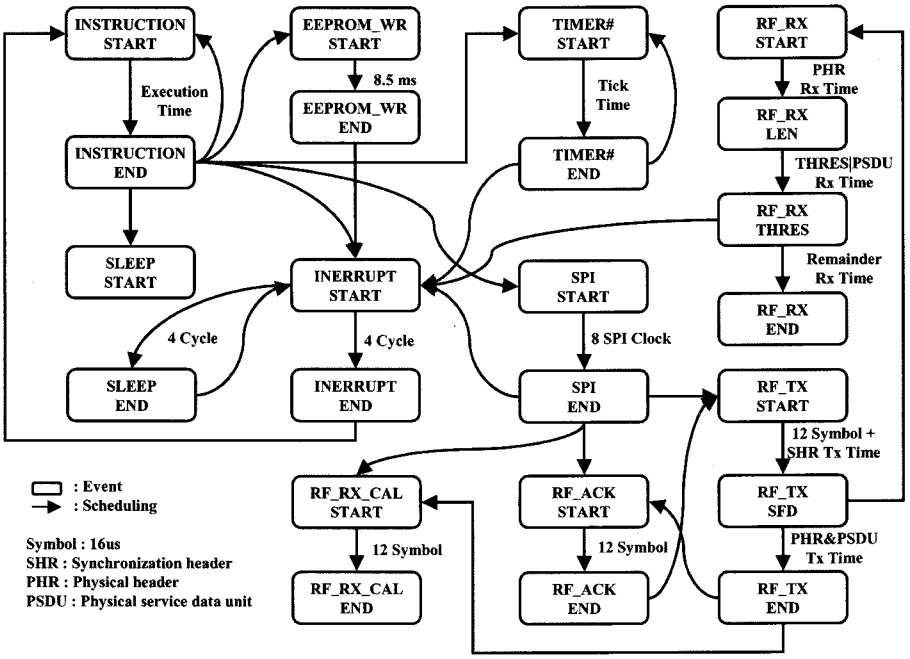


그림 6 PASENS에서 사건들 간의 스케줄링 관계

실행이미지에서 PC가 가리키는 위치의 명령어 하나를 읽어서 해독하고 해당 명령어의 실행 시간 후에 INSTRUCTION_END 사건이 수행되도록 스케줄 한다. 만약 MCU의 시스템 클록이 8 ㎒이고, 명령어가 그림 4의 "ADD" 명령어라면 125 ns 후에 INSTRUCTION_END 사건이 수행되도록 스케줄 한다.

• INSTRUCTION_END 루틴

명령어의 실행이 실제 수행되는 루틴으로서, 해당 명령어의 실행 결과에 따라 변경된 상태를 가상 센서노드의 각 레지스터에 반영한다. 이 루틴에서는 명령어의 실행 결과나 센서노드의 상태에 따라서 여섯 가지의 다른 사건들을 스케줄 한다. 첫 번째는 "sleep" 명령어가 실행된 후에는 센서노드를 슬립 상태로 전환시키기 위해 SLEEP_START 사건을 스케줄 한다. 두 번째는 명령어의 실행 결과로서 EEPROM 쓰기가 발생할 경우에는 EEPROM_WR_START 사건을 스케줄 한다. 세 번째는 명령어의 실행 결과로서 타이머/카운터 제어 레지스터(timer/counter control register: TCCR)의 특정 비트가 변경되었을 경우에 타이머를 동작시키기 위해 TIMER_START 사건을 스케줄 한다. 네 번째는 명령어가 실행되는 동안 인터럽트가 발생했다면 인터럽트 루틴을 실행하기 위해서 INTERRUPT_START 사건을 스케줄 한다. 다섯 번째는 명령어 실행 결과로서 RF 모듈을 이용하기 위해 SPI를 통한 데이타의 전송이 발생하게 되면 SPI_START 사건을 스케줄 한다. 마지막으로, PC를 증가시킨 후에 INSTRUCTION_START 사건을 스케줄 하여 다음 명령어를 실행할 수 있도록 한다.

• RF_TX_START 루틴

RF 송신을 시작하는 루틴으로서, 8 심벌(128 ㎲) 또는 12 심벌(192 ㎲) 시간 지연과 SHR(synchronization header, preamble) 전송 시간 후에 RF_TX_SFD 사건이 발생하도록 스케줄 한다. CC2420에서 대역폭은 250 Kbps이기 때문에 1 비트의 전송 시간은 4 ㎲이다.

• RF_TX_SFD 루틴

송신 노드에서 SHR까지 송신한 후에 실행되는 루틴으로, PASENS에서는 이 시점에서 송신 노드의 송신 버퍼에 있는 패킷 내용을 수신 노드의 수신 버퍼로 복사한다. 그림 7에서 보는 바와 같이 패킷의 내용이 수신되는 시간에 따라 수신 노드의 CC2420 핀에 변화가 발생하기 때문에, 이 때 RF 수신 동작을 수행하기 위하여 수신 노드에 RF_RX_START 사건을 스케줄 한다. 이와 동시에 송신 노드에는 송신 완료 동작을 수행하기 위한 RF_TX_END 사건이 송신 패킷의 PHR(physical header)과 PSDU(physical service data unit)의 전송이 끝나는 시간 후에 발생하도록 스케줄 한다. 만약 전송될 데이타의 크기가 10 Byte라면 320 ㎲ 시간 후에 RF_TX_END 사건이 발생한다. 수신 노드에서 SFD 신호는

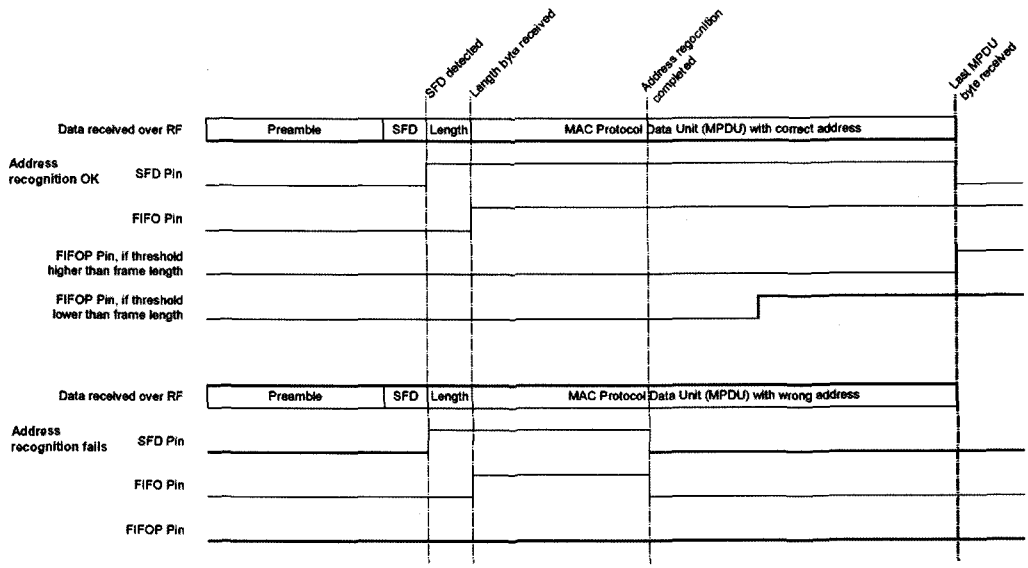


그림 7 CC2420의 RF 수신 동작

RF 메시지를 수신중임을 나타내고, FIFOP 신호는 RF 수신을 MCU에게 알리는 인터럽트 신호로 사용된다.

• RF_RX_START 루틴

RF 수신 노드에서 패킷을 SHR까지 수신한 시점에 발생하는 동작을 시뮬레이션 하는 루틴이다. 이 시점에 MCU로 입력되는 CC2420의 SFD(start-of-frame delimiter) 핀의 값이 1로 바뀌게 되고, 이때부터 수신 노드의 MCU는 패킷이 수신되었다는 것을 알 수 있다. 수신 동작의 다음 처리를 위하여 PHR이 수신되는 시점에 RF_RX_LEN 사건이 발생하도록 스케줄 한다.

3.5 병렬 이산 사건 시뮬레이션

PDES는 이산 사건 시뮬레이션에서 시뮬레이션 시간을 단축시키기 위하여 사용된다. PDES에서 사건들은 논리 프로세스(logical process: 이하 LP라 함)들로 나누어져 처리되는데, 각 LP는 VT를 증가시키면서 자신의 사건 큐에 할당된 사건들을 시뮬레이션 한다. 만약 각 LP에 분산되어 시뮬레이션 되는 사건들이 하나의 LP에서 시뮬레이션 되는 경우와 동일한 시간 순서대로 처리되지 않았다면, 인과성 오류(causality error)가 발생했다고 한다. PDES의 일반적인 방법에는 동기식 시뮬레이션(synchronous simulation)과 보수적 시뮬레이션(conservative simulation), 그리고 낙관적 시뮬레이션(optimistic simulation)이 있다. 보수적 시뮬레이션과 낙관적 시뮬레이션은 비동기식 시뮬레이션(asynchronous simulation) 방법으로서 프로세서들 간의 지역 가상시간(local virtual time: 이하 LVT라 함)이 일치하지 않는 반면에, 동기식 시뮬레이션에서는 프로세서들 간의 LVT가 항상 일치한다[27].

최적 동기식 PDES 방법은 전체 가상시간(global virtual time: 이하 GVT라 함)를 유지하면서 병렬 시뮬레이션을 전체적으로 제어하는 하나의 제어 프로세스(control process: 이하 CP라 함)와 센서노드들을 분할하여 시뮬레이션을 수행하는 여러 개의 LP들로 구성된다. 이 방법은 CP와 각 LP를 이더넷으로 연결된 각각의 컴퓨터가 하나씩 담당하여 처리하는 중앙집중식 구조이다. 이와 같이 이더넷에 연결된 PC들을 사용하는 컴퓨팅 환경은 최적 동기식 PDES 방법을 저렴한 비용으로 쉽게 구축할 수 있게 한다.

최적 동기식 PDES 방법은 동기식 시뮬레이션에 기반을 두지만 비동기식 시뮬레이션의 특징도 가지고 있는 방법이다. 동기식 시뮬레이션에 기반을 둔 이유는 명령어 시뮬레이션이 동기식 시뮬레이션을 주로 사용하는 회로 시뮬레이션(logic simulation)처럼 기본 계산 단위가 매우 작아서 사건의 동작을 처리하는 시간이 매우 짧기 때문이다. 그러나 회로 시뮬레이션과는 달리 WSN 시뮬레이션은 RF 통신 동작 외에는 센서노드 별로 독립적으로 동작하기 때문에 센서노드 단위로 문제 분할이 용이하다. 또한 어떤 센서노드에서 RF 송신이 시작된 후부터 다른 센서노드에서 RF 수신에 따른 동작이 발생할 때까지는 일정한 시간 지연이 존재한다. 따라서 최적 동기식 PDES 방법은 그 지연 시간 간격 단위로 동기식 시뮬레이션을 진행하고, 그 시간 간격 내에서 RF 송신 동작이 발생하면 다음 시간 간격 동안에 그것을 수신하는 모든 센서노드에서 RF 수신 동작이 발생

```
/* Δt : A time interval of GVT
   End_Time : End of simulated time */
GVT = 0;
do {
   GVT += Δt;
   if (GVT > End_Time)
     GVT = End_Time;
   Broadcast GVT to all LPs;
   Wait until receiving completion messages from all LPs;
} while (GVT < End_Time);
```

(a) CP

```
/* E_i : ith event
   LVT_i : The occurrence time of E_i */
while (1) {
   Wait for an GVT from CP;
   while (1) {
      Remove an event E_i with the earliest LVT_i from the event queue;
      LVT = LVT_i;
      if (LVT > GVT) {
         Insert E_i into an event queue;
         LVT = GVT;
         break;
      }
      Execute the event routine of E_i;
      if (RF transmission occurs by E_i)
         Broadcast the RF message to other LPs;
   }
   Insert an event to process the received message into the event queue;
   Send a completion message to CP;
}
```

(b) LP

그림 8 최적 동기식 PDES 알고리즘

하도록 사건을 스케줄 함으로써, 분배된 모든 사건이 시간 순서대로 수행되도록 한다. 결과적으로 최적 동기식 시뮬레이션 방법은 인과율 제약을 만족시키는 일정 시간 간격으로 GVT를 증가시키면서 동기식 시뮬레이션을 수행하지만, 그 시간 간격 내에서는 각 LP가 LVT를 증가시키면서 비동기식 시뮬레이션을 수행하게 된다.

최적 동기식 PDES에서 CP는 배리어 동기화 방법을 이용하여 GVT를 유지하면서 LP들을 제어하여 병렬 시뮬레이션을 수행한다. 시뮬레이션이 시작되면 CP는 시뮬레이션이 종료되는 VT까지 인과성 제약을 만족시키는 일정 시간 간격(Δt)으로 GVT를 증가시킨다. 이 GVT는 멀티캐스팅을 사용하여 시뮬레이션에 참여하는 모든 LP들에게 전달된다. GVT를 받은 LP들은 GVT까지 LVT를 증가시키면서 각각 시뮬레이션을 진행한다. LP는 사건 큐에서 가장 빨리 발생될 사건을 꺼내고, 그 사건이 발생되는 시간으로 LVT를 갱신한다. 사건 큐에서 꺼낸 사건의 LVT가 GVT 이하일 경우에는 사건 루틴을 실행하며, 실행 중에 RF 송신이 발생하면 다른 LP들에게 RF 메시지를 멀티캐스팅 한다. RF 메시지를 받은 LP는 이 메시지를 처리하기 위한 사건을 스케줄 한다. 만약 사건 큐에서 꺼낸 사건의 갱신된 LVT가 GVT보다 크다면, 꺼낸 사건을 다시 사건 큐에 삽입하고 LVT는 GVT로 갱신한다. 그리고 LP는 GVT까지 시뮬레이션을 완료했다는 메시지를 전송한 후에 다음 GVT를 수신할 때까지 대기한다. CP는 모든 LP들로부터 시뮬레이션 완료 메시지를 수신하면 GVT를 증가시킨다. 그림 8은 CP와 LP의 최적 동기식 PDES 알고리즘을 보여준다.

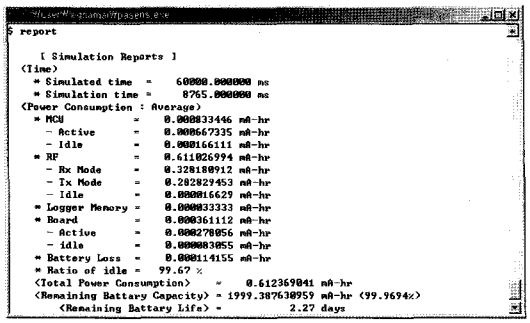
```
$ add 2
[Msg] ID of created node :
      0    1
$ load CntToRfm.srec
[Msg] (CntToRfm.srec) loaded in node 0's flash memory.
$ nch 1
[Msg] Current node is changed : 0 => 1
$ load RfmToLeds.srec
[Msg] (RfmToLeds.srec) loaded in node 1's flash memory.
$ siminit
[Msg] Environment values is loaded from 'pasens.cfg'
[Msg] Initialize simulation parameter.
$ simrun 60000
[Msg] Node-0 : RF-Tx, Time=  249.371474 ms, TxCnt=14, FREQ=2405
[Msg] Node-1 : RF-Rx (1 <- 0), Time=  249.755474 ms
      Packet : 0f 01 00 01 ff ff ff ff 04 7d 01 00 01 00
[Msg] Node-1 : LED = __R, Time=  249.958110 ms
[Msg] Node-0 : RF-Tx, Time=  489.128825 ms, TxCnt=14, FREQ=2405
[Msg] Node-1 : RF-Rx (1 <- 0), Time=  489.512825 ms
      Packet : 0f 01 00 02 ff ff ff ff 04 7d 02 00 01 00
[Msg] Node-1 : LED = ___, Time=  489.715733 ms
[Msg] Node-1 : LED = _G_, Time=  489.717767 ms
[Msg] Node-0 : RF-Tx, Time=  732.003716 ms, TxCnt=14, FREQ=2405
[Msg] Node-1 : RF-Rx (1 <- 0), Time=  732.387716 ms
      Packet : 0f 01 00 03 ff ff ff ff 04 7d 03 00 01 00
[Msg] Node-1 : LED = _GR, Time=  732.590353 ms
[Msg] Node-0 : RF-Tx, Time=  974.564345 ms, TxCnt=14, FREQ=2405
```

그림 9 실행시간 예측 예

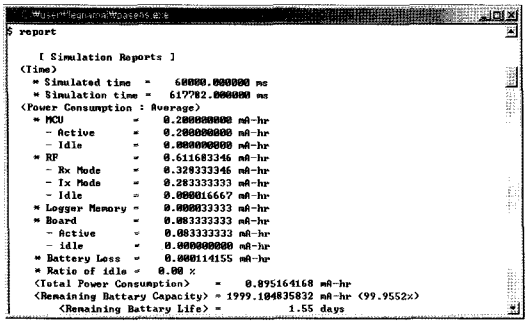
4. 실 험

첫 번째 실험은 PASENS의 실행시간 및 전력소모량의 예측 기능을 검증하기 위하여, 두 개의 센서노드에 TinyOS[3] 1.1.7에 포함되어 있는 예제들인 "CntToRfm"와 "RfmToLeds"를 각각 적재하고 60초의 VT 동안 시뮬레이션을 진행하였다. "CntToRfm"은 4 Hz 간격으로 증가된 카운터 값을 RF 통신으로 전송하는 예제이고, "RfmToLeds"는 RF 통신으로 수신된 카운터 값의 하위 세 비트를 LED로 출력하는 예제이다. 이 실험에서 시뮬레이션 대상인 센서노드는 MICAz 형식을 가지는 CrossBow 사의 MPR2400 보드[22]로 설정하였으며, 이 센서보드의 MCU 시스템 클록은 7.37 ㎒이다. 시뮬레이션을 수행한 PC의 프로세서는 1.7 ㎓ 펜티엄 4이고, 주기억장치의 용량은 256 MB이며, 운영체제는 MS 윈도우즈 XP이다.

그림 9의 실험결과는 약 0.25초 간격으로 0번 센서노드에서 1번 센서노드로 카운터 값이 전송되고, 전송된 카운터 값의 하위 3비트 값이 1번 센서노드의 LED로 표시되는 것을 보여준다. 이 결과는 PASENS가 WSN 응용프로그램의 동작을 정확히 시뮬레이션하고, 실행시간을 예측할 수 있다는 것을 의미한다. 실행화면에서 시간 단위는 ms이며, "LED = " 뒤에 있는 'Y', 'G', 'R'은 각각 황색, 녹색, 적색 LED의 점등을 나타내고, "RF-Rx" 뒤에 있는 내용은 수신된 패킷의 송신 센서노드 ID와 내용을 보여준다. "FREQ= " 뒤에 나오는 값은 RF 송신 주파수를 나타내고, 패킷의 내용 중에 11번째 바이트 부분이 카운터 값이다. 그림 9와 같이 PASENS는 대소문자를 구별하는 쉘 명령어에 의해 동작하며, "help" 명령어를 사용하면 지원되는 쉘 명령어의 종류와 사용법을 보여준다. 대부분의 시뮬레이션은 "add", "load", "siminit", 그리고 "simrun" 명령어의 순서에 의해서 이



(a)



(b)

그림 10 전력소모량 예측 예

루어진다. 즉, 가상 센서노드를 생성하고, 생성된 노드에 실행이미지를 적재한 후에 초기화 작업을 수행하고 시뮬레이션을 수행한다.

그림 10(a)는 첫 번째 실험에서 센서노드 내부의 동작시간을 기반으로 평균 전력소모량을 예측한 결과를 보여주며, 기준 소모 전력량은 CrossBow사에서 제공하는 전력소모량[22]을 사용하였다. "MCU", "RF", "Logger Memory"는 센서보드의 각 부품에서 소모된 전력량을 나타내고, "Board"는 그 외에 소모된 전력량을 나타내며, "Battery Loss"는 건전지에서 자연적으로 방전되어 소모되는 전력량을 나타낸다. 이 예제들은 실행시간의 99.67%가 슬립 상태이며, 계속 실행시켰을 경우에 2000 mA-hr의 건전지로 2.27일 동안 동작이 가능하다. 반면에 그림 10(b)는 슬립 상태가 없도록 예제를 수정하여 동일한 방법으로 시뮬레이션 하였을 때의 전력소모량을 보여주고 있는데, 이 경우에는 앞으로 1.55일 동안 동작이 가능하다. 이 결과는 WSN 응용프로그램에 따라 WSN의 수명이 어느 정도일지 예측할 때에 PASENS가 유용하다는 것을 나타낸다.

MCU 사이클 단위의 시뮬레이션 정밀도를 갖는 PASENS는 그림 11과 같이 정밀한 수준으로 WSN 응용프로그램을 디버깅할 수 있다. 특히 MCU와 RF 모듈에 대하여 레지스터 수준의 자세한 정보를 사용자에게

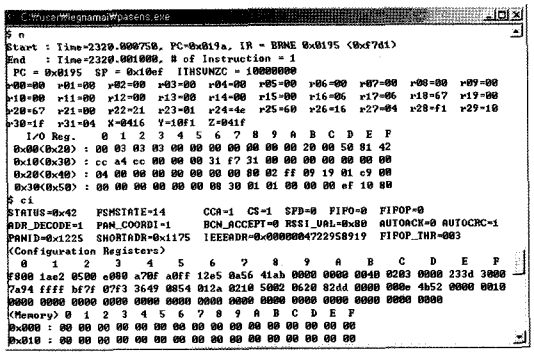


그림 11 WSN 응용프로그램의 디버깅 화면

모두 제공하는 WSN 시뮬레이터로는 현재 PASENS가 유일하다. 실행화면은 MCU 시스템 클록이 8 ㎒인 Octacomm사의 NANO-24 보드[23]에서 한국전자통신연구원(ETRI)의 NANO-Q+[28]에 포함되어 있는 'detect_net4' 예제를 디버깅하는 상황을 보여주고 있다. 여기서 사용된 "n" 명령어는 한 기계명령어를 실행한 후에 MCU 내부 레지스터들의 내용을 보여주고, "ci" 명령어는 CC2420 내부 레지스터들의 내용과 RF 모듈에 관한 정보들을 보여준다. "CCA", "CS", "SFD", "FIFO", "FIFOP"는 CC2420 외부 핀들의 현재 값을 나타내고, "PANID", "SHORTADR", "IEEEADR"은 IEEE 802.15.4의 네트워크 주소를 나타낸다. 디버깅 상황은 PC가 0x019a 번지일 때, PC가 가리키는 플래시 기억장치의 위치에서 기계명령어 "0×f7d1"을 인출하여 "brne 0×0195" 명령어로 해석하여 실행한 것을 나타낸다. 실행결과로서 PC는 0x0195로 변경이 되었으며, VT는 2 사이클의 MCU 클록에 해당되는 250 ㎱가 진행되었음을 알 수 있다. 만약 확장 I/O 레지스터의 내용을 보길 원한다면 "xior" 쉘 명령어를 사용할 수 있으며, 내부 또는 외부 SRAM의 내용을 보길 원한다면 "m" 쉘 명령어를 사용할 수 있다.

PASENS는 쉘 명령어들을 포함하는 일괄파일(batch file)을 사용하여 온도 변화나 센서노드 고장과 같은 다양한 시나리오의 WSN의 동작을 시뮬레이션 할 수 있다. 그림 12는 NANO-Q+에 포함되어 있는 'temperature' 예제를 사용하여 40초의 VT동안 온도 변화와 센서노드 고장 및 복구를 'etri_temp.sh' 일괄파일로 시뮬레이션 한 결과를 보여준다. 'temperature'는 모든 LED를 켜고 끈 후에 10초 간격으로 온도를 측정하여 직렬 모니터(serial monitor)로 그 값을 출력하는 예제이다. 'etri_temp.sh' 일괄파일은 VT가 10초일 때에 온도가 31.5도로 변경되고, 20초일 때에 센서노드가 고장이 발생하며, 30초일 때에 센서노드가 복구되어 다시 동작하는 시나리오를 수행한다. 그림 12의 결과에서 보면, VT

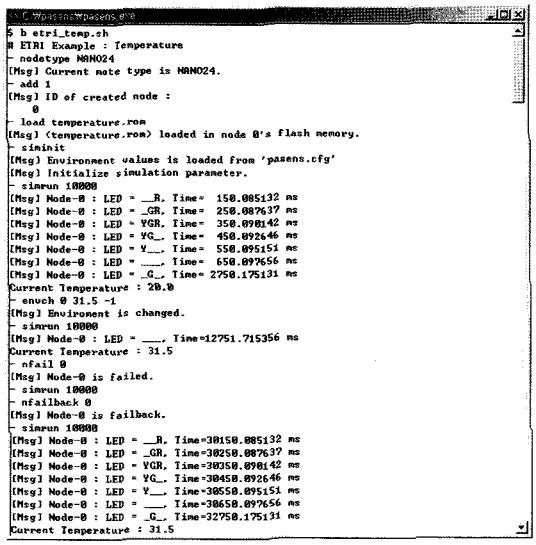


그림 12 일괄파일을 사용한 시나리오 시뮬레이션

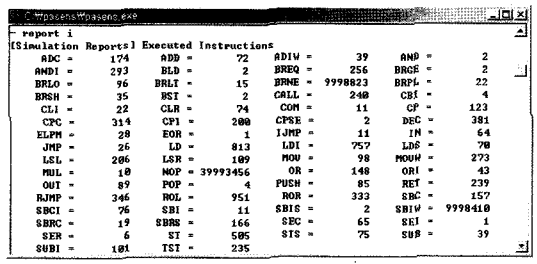


그림 13 실행된 명령어들의 수

가 2.75초일 때에 "C:\pasens.cfg"에 설정된 초기 온도인 20도가 직렬 모니터로 출력되고, 12.75초일 때에는 변경된 온도인 31.5도가 출력된다. 그리고 VT가 22.75초일 때에는 센서노드가 고장 상태이기 때문에 온도가 출력되지 않으며, VT가 32.75일 때에는 센서노드의 고장이 복구되어 온도가 출력되는 것을 볼 수 있다. 이와 같이 PASENS는 쉘 명령어들을 조합하여 다양한 시나리오를 일괄파일로 작성하여 실험할 수 있다.

그림 13은 'temperature' 실험에서 실행된 명령어들을 종류별로 계수한 결과를 보여준다. 이 결과를 통하여 컴파일러의 경향을 분석할 수 있으며, 각 명령어가 실행될 때의 전력소모량을 실측한 값을 사용한다면 더 정확한 전력소모량을 예측할 수 있다.

마지막 실험은 PASENS의 시뮬레이션 시간을 다른 WSN 시뮬레이터들과 비교하기 위하여 Avrora[4]의 실험과 유사한 환경에서 진행하였다. 동일한 환경에서 실험을 진행하지 않은 이유는 다른 WSN 시뮬레이터들이 유닉스 기반에서 구현된 시뮬레이터이기 때문이다. Avrora의 실험에서는 3.06 ㎓의 듀얼 Xeon 프로세서와 4 GB

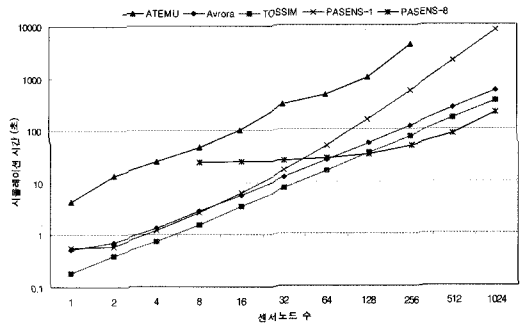


그림 14 PASENS와 다른 WSN 시뮬레이터들과의 시뮬레이션 시간 비교

의 주기억장치, 그리고 리눅스 2.4.20 운영체제의 컴퓨터를 사용한 반면에, 본 실험에서는 2.8 ㎓ 펜티엄 4 프로세서와 1 GB의 주기억장치, 그리고 MS 윈도우즈 XP 운영체제의 컴퓨터를 사용하였다. 그리고 센서노드의 수를 1개부터 1,024개까지 증가시키면서, 절반의 센서노드들에는 "CntToRfm"를, 나머지 절반의 센서노드들에는 "RfmToLeds"를 나누어 적재시키고 10초의 VT 동안 시뮬레이션 하였다. 그림 14는 PASENS의 실험 결과를 Avrora[4]에서 제시한 실험 결과와 함께 보여주는 것으로서, PASENS-1은 단일 시뮬레이션 결과를 나타내고 PASENS-8은 8개의 LP를 사용하여 병렬 시뮬레이션을 한 결과이다.

그림 14의 결과에서 보면 PASENS-1의 시뮬레이션 시간은 ATEMU[20]보다는 짧지만 Avrora[4]나 TOSSIM[2]에 비해서는 더 길다는 것을 알 수 있다. 시뮬레이션 정밀도가 낮은 TOSSIM은 TinyOS의 컴포넌트 단위로 이산 사건 시뮬레이션을 수행하기 때문에 시뮬레이션 시간이 가장 짧고, ATEMU는 슬립 상태에서도 시스템 클록 단위로 시뮬레이션을 진행하기 때문에 시뮬레이션 시간이 가장 길다. 그리고 Avrora가 PASENS-1에 비하여 시뮬레이션 시간이 짧은 이유는, 센서노드를 하나의 쓰레드로 동작시키는 Avrora를 다중 쓰레드 동작이 용이한 하이퍼 쓰레드 기술(hyper-threading technology)이 적용된 듀얼 Xeon 프로세서를 사용하여 실험하였기 때문이다. 즉, Avrora는 4개의 프로세서에서 실험한 결과인 반면에, PASENS-1은 한 프로세서에서 실험한 결과라고 볼 수 있다. 8개의 LP를 사용한 PASENS-8은 센서노드의 수가 적을 때에는 병렬 시뮬레이션의 오버헤드로 인하여 TOSSIM이나 Avrora에 비하여 시뮬레이션 시간이 길었다. 그러나 센서노드의 수가 128개 이상으로 증가하면 PASENS-8이 가장 짧은 시뮬레이션 시간을 나타내었고, 센서노드의 수가 1024개일 경우에는 PASENS-8의 시뮬레이션 시간이 TOSSIM에 비해서는 1.6배, 그리고 Avrora에 비해서는

2.6배 빠른 결과를 얻을 수 있었다.

5. 결 론

본 연구에서는 유비쿼터스 컴퓨팅의 인프라가 되는 WSN을 가능한 한 정확하게 시뮬레이션 하기 위하여 명령어 수준의 이산 사건 시뮬레이션 방법과 최적 동기식 PDES 방법을 사용하여 병렬 시뮬레이터인 PASENS를 개발하였다. 명령어 수준 시뮬레이션은 센서노드의 시스템 클록 수준으로 시뮬레이션의 정밀도를 높임으로써, 센서노드의 동작과 실행시간 및 전력소모량을 실제와 거의 근접한 수준으로 예측할 수 있게 한다. 또한 이 방법은 실제 센서노드에 적재되는 실행이미지를 시뮬레이션 작업부하로 사용하여 기존의 많은 WSN 시뮬레이터들이 가진 소프트웨어 의존성 문제를 해결하였다. 그러나 명령어 시뮬레이션은 정밀도가 높기 때문에 시뮬레이션 시간이 현저히 길어져서 시뮬레이션 할 수 있는 센서노드의 수에 제한이 있다는 문제점이 있다. 최적 동기식 PDES 방법은 그러한 문제점을 해결하여 시뮬레이션 시간을 크게 단축시킴으로써 대규모의 WSN을 시뮬레이션 할 수 있게 한다.

PASENS는 센서보드 내의 모듈 별로 동작시간을 사용하여 전력소모량을 예측할 수 있고, 실행된 명령어들을 종류별로 계수할 수 있다. 따라서 모듈 또는 명령어 별로 실측한 전력소모량을 사용하면, PASENS는 더 정확한 전력소모량을 예측할 수 있다. 또한 PASENS는 MCU와 RF 트랜시버의 내부 모듈들의 동작을 레지스터 수준으로 디버깅할 수 있고, 온도의 변화나 센서노드의 고장 및 복구와 같은 다양한 시나리오의 WSN의 동작을 시뮬레이션 할 수 있다. PASENS 프로그램의 실행 파일과 프로그램 소스는 'http://caeagle.yonsei.ac.kr/pasens/' 에서 제공된다.

참 고 문 헌

[1] 김방현, 김태규, 정용덕, 김종현, "실행시간 및 전력소모량 추정이 가능한 센서 네트워크 시뮬레이터의 개발," 한국시뮬레이션학회 논문지, 15(1): 35-42, 2006년 3월.

[2] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications," In Proceedings of 1st ACM Conference on Embedded Networked Sensor Systems, 2003.

[3] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D.E. Culler, and K.S. Pister, "System Architecture Directions for Networked Sensors," In Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems, 2000.

[4] B.L. Titzer, D.K. Lee, and J. Palsberg, "Avrora: Scalable Sensor Network Simulation with Precise Timing," In Proceedings of The 4th IEEE International Symposium on Information Processing in Sensor Networks, 2005.

[5] The Network Simulator, http://www.isi.edu/nsnam/ns/.

[6] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable And Robust Communication Paradigm For Sensor Networks," In Proceedings of the International Conference on Mobile Computing and Networking, Aug. 2000.

[7] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, "GHT: A Geographic Hash Table for Data-Centric Storage," In Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications, 2002.

[8] The Monarch Project, http://www.monarch.cs.rice.edu/.

[9] S. Park, A. Savvides, and M.B. Srivastava, "SensorSim: A Simulation Framework for Sensor Networks," In Proceedings of the 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems, 2000.

[10] The NEST Project, http://webs.cs.berkeley.edu/.

[11] G. Simon, P. Volgyesi, M. Maroti, and A. Ledeczi, "Simulation-based Optimization of Communication Protocols for Large-scale Wireless Sensor Networks," In Proceedings of the IEEE Aerospace Conference, March 2003.

[12] A. Ledeczi, M. Maroti, and I. Bartok, "Simple Nest Application Simulator," Draft, Institute for Software Integrated Systems, Vanderbilt University, October 2001.

[13] L.F. Perrone and D.M. Nicol, "A Scalable Simulator for TinyOS Applications," In Proceedings of the 2002 Winter Simulation Conference, 2002.

[14] J. Liu, D. Nicol, F. Perrone, and M. Liljenstam, C. Elliot, and D. Pearson. "Simulation Modeling of Large-Scale Ad-hoc Sensor Networks," In Proceedings of European Interoperability Workshop 2001, June 2001.

[15] V. Shnayder, M. Hempstead, B. Chen, G.W. Allen, and M. Welsh, "Simulating The Power Consumption of Large-Scale Sensor Network Applications," In Proceedings of The 2nd ACM International Conference on Embedded Networked Sensor Systems, Nov. 2004.

[16] M. Demmer, P. Levis, A. Joki, E. Brewerr, and D. Culler, "Tython: a Dynamic Simulation Environment for Sensor Networks," Technical Report, CSD-05-1372, UC Berkeley, Feb. 2005.

[17] The Python Programming Language, http://www.python.org/.

[18] S. Sundresh, W. Kim, and G. Agha, "SENS: A Sensor, Environment and Network Simulator," The 37th Annual Simulation Symposium, April

2004.
[19] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin, "EmStar: a Software Environment for Developing and Deploying Wireless Sensor Networks," In Proceedings of USENIX Technical Conference, 2004.
[20] J. Polley, D. Blazakis, J. McGee, D. Rusk, J.S. Baras, and M. Karir, "ATEMU: A Fine-grained Sensor Network Simulator," In Proceedings of The First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004.
[21] M. H. MacDougall, Simulating Computer Systems: Techniques and Tools, MIT Press, July 1987.
[22] CrossBow, MPR/MIB Users Manual, April 2005.
[23] Octacomm, http://www.octacomm.net/.
[24] Atmel, ATmega128(L) Complete, March 2006.
[25] ChipCon, SmartRF CC2420 Preliminary Datasheet 1.2, June 2004.
[26] A. Alan B. Pritsker, Jean J. O'Reilly, and David K. LaVal, Simulation with Visual SLAM and AweSim, John Wiley & Sons, March 1999.
[27] A. Ferscha, Parallel and Distributed Simulation of Discrete Event Systems, In Handbook of Parallel and Distributed Computing, McGraw-Hill, 1995.
[28] NANO-Q+, http://qplus.or.kr/.

김 방 현
1996년 연세대학교 전산학과 이학사 2001년 연세대학교 전산학과 이학석사 2007년 연세대학교 컴퓨터정보통신공학부 이학박사. 관심분야는 센서 네트워크, 병렬처리, 시뮬레이션

김 종 현
1976년 연세대학교 전기공학과 공학사 1981년 연세대학교 전기공학과 공학석사 1988년 Arizona State University 전기 및 컴퓨터공학과 Ph.D. 1976년~1982년 국방과학연구소 연구원. 1988년~1990년 한국전자통신연구소 실장 역임. 1990년~현재 연세대학교 컴퓨터정보통신공학부 교수. 1996년 Oregon State University 전산학과 방문교수. 2003년 Florida State University 전산학과 방문교수. 관심분야는 컴퓨터구조, 병렬처리, 시뮬레이션, 센서 네트워크