네트워크 기반의 소형 유비쿼터스 시스템의 개발

(Designing of Network based Tiny Ubiquitous Networked Systems)

황 광 일 † 엄 두 섭 ††
(Kwang-il Hwang) (Doo-seop Eom)

요 약 본 논문에서 우리는 ELOS(Embedded Lightweight Operating System)라 불리는 이벤트 기반의 운영체제와 멀티홉 에드혹 라우팅 프로토콜로 구성된 네트워크 기반의 소형 실시간 시스템의 구조를 제시한다. 효율적인 실시간 프로세싱을 위하여 보장된 시간 슬롯을 가진 조건적 선점형 FCFS 스케줄러가 개발되었다. 보다 정교한 네트워크 구성을 위하여 무선 에이전트 노드를 통한 반자동 구성(semi-auto configuration) 방식을 사용한다. 개발된 소프트웨어 시스템은 자체 개발한 소형 하드웨어 프로토타입에서 구현되었다. 또한, 제안된 시스템의 성능을 평가하기 위해서, 우리는 유비쿼터스 네트워크 테스트 베드를 개발했고, 다양한 환경에서의 실험이 이루어 졌다. 실험 결과를 통하여 제안된 ELOS 시스템은 실시간 제약을 가진 네트워크 기반의 소형 유비쿼터스 시스템에 상당히 알맞은 시스템이라는 것을 확인한다.

키워드 : 임베디드 시스템, 운영체제, 실시간 시스템, 유비쿼터스 컴퓨팅, 센서 네트워크

Abstract In this paper, we present a network-oriented lightweight real-time system, which is composed of an event-driven operating system called the Embedded Lightweight Operating System (ELOS) and a generic multi hop ad hoc routing protocol suite. In the ELOS, a conditional preemptive FCFS scheduling method with a guaranteed time slot is designed for efficient real-time processing. For more elaborate configurations, we reinforce fault tolerance by supplementing semi-auto configuration using wireless agent nodes. The developed hardware platform is also introduced, which is a scalable prototype constructed using off-the-shelf components. In addition, in order to evaluate the performance of the proposed system, we developed a ubiquitous network test-bed on which several experiments with respect to various environments are conducted. The results show that the ELOS is considerably favorable for tiny ubiquitous networked systems with real-time constraints.

Key words : Embedded Systems, Operating Systems, Real-time Systems, Ubiquitous Computing

1. 서 론

Ubiquitous networks are emerging as a next network paradigm in which infrastructure and services are available in any format, at any time or location. This exciting new paradigm is the result of recent research and technological advances in wireless and sensor networks, distributed systems, micro-electro-mechanical (MEMS) and low cost CMOS technology. Ubiquitous networks also enabled development of a variety of applications, from home networks for environmental monitoring to home, social or industrial network applications. To provide seamless ubiquitous services, it is important that hundreds or thousands of ubiquitous nodes collaborate in a distributed ad hoc manner. Such nodes should have a scalable architecture for supporting diverse applications.

Ubiquitous nodes, or sensor nodes, consist of extremely limited resources, more specifically, processing, memory, and power. In addition to such deteriorate resources, a network-oriented ubiquitous system has different characteristics from general purpose networked embedded system. In ubiquitous networks, it is very frequent that several different

† 정 회 원 : 시립인천전문대학 컴퓨터제어과 교수
brightday@final.korea.ac.kr
†† 정 회 원 : 고려대학교 전자공학과 교수
eomds@final.korea.ac.kr
논문접수 : 2006년 4월 7일
심사완료 : 2007년 5월 22일

flows of data occur on the network, simultaneously. These traffics are not always but intermittent since they generate a traffic only when they sense environmental variations or respond to query of a user. In addition, most of ubiquitous applications have real-time constraints. In particular, it becomes more significant requirement under ubiquitous environment where most nodes should be cooperatively synchronized for autonomous multi hop communication.

To cope well with such characteristics of ubiquitous networked systems, in this paper, a network-oriented lightweight real-time system, which is composed of an event-driven operating system called the Embedded Lightweight Operating System (ELOS) and a generic multi hop ad hoc routing protocol suite.

In the following section, traditional researches related to ubiquitous networked systems are introduced. The proposed system architecture contrasted with traditional works is presented in section Ⅲ. Section Ⅳ introduces the developed hardware platform, which is a scalable prototype constructed using off-the-shelf components. In section Ⅴ, the performance of the proposed system, which is implemented in a ubiquitous network test-bed constructed in our building, is evaluated from the various viewpoints. Section Ⅵ concludes this paper.

2. Related work

Practical real-time ubiquitous applications are successfully accomplished by a harmonious combination of various technologies, such as embedded hardware platform, system software including operating system, and network technology based on ad hoc sensor networks.

First, several hardware platforms, which are composed using commercial off-the-shelf components, have been developed for ubiquitous sensor networks, as in [1-6]. The platform [1] using StrongARM processor as an MCU has achieved high performance processing. However, because of expensive production cost per node, it is not desirable to be massively deployed in a number of nodes. Instead it is more desirable to be used for minor limited systems in which high processing ability is required. The other platforms [2-6] which use 8-bit micro controller, such as an Atmega 8bit AVR RISC processor or 8051 series, fill the requirements of being low cost, but requires additional efforts to be extended to diverse and flexible systems, since they only support consistent I/O components and RF modems.

In parallel with the development of hardware prototypes, various embedded software platforms [7-12] have been developed. The researches usually focus on the development of embedded operating systems to efficiently manage the limited resources. Operating systems [7-10] for traditional real-time embedded systems allow some components to be added or removed according to their needs. However, the size and functions of the operating systems are still much more than what is required for lightweight real-time ubiquitous networked systems. In order to operate on low performance microcontrollers, compact real-time operating systems [11,12] have been introduced. In the systems, resource allocation and scheduling are mainly targeted for control system applications, not communications. This feature is not suitable for the real-time ubiquitous system, in that most tasks are tightly related to communication events. TinyOS [6] is the most popular operating system developed on the Berkley Mote platform for sensor networks. The event-driven operating system adopts a LCFS scheduling method. However, in a real-time ubiquitous environment where time-critical applications exist, since context switching by new tasks is required frequently, multiple tasks can create data-race conditions and eventually the service time of time-critical tasks will not guaranteed.

In a ubiquitous network, it is necessary to disseminate information to certain specific nodes, such as a base station or sink node, through multi-hop ad hoc routing technology. Several primitive ad hoc routing algorithms [14] have been already introduced. In addition, sensor network routing protocols [15] perform data aggregation coupled with data-centric routing scheme in more limited resource environments. The Zigbee alliance are trying to accommodate a variety of routing topologies, such as cluster tree, mesh, and star, by

extending low data rate WPAN [16]. To accommodate these various ad hoc routing protocols in ubiquitous environments, our developed system, ELOS, supports various RF modems and several routing protocols modified on the basis of traditional routing protocols, such as flooding, tree-based routing, and mesh supported routing protocols.

While existing researches are not completely suitable to be integrated into a generic real-time ubiquitous networked system due to several problems in flexibility and guaranteeing a real-time service, the proposed ELOS system is designed to conform well to a scalable real-time ubiquitous system.

3. The proposed real-time system software architecture

The ubiquitous system is different from traditional general purpose real-time embedded systems in that each system conducts an application specific operation. Figure.1 describes the proposed real-time software architecture for the ubiquitous real-time networked system. The software architecture is largely divided into three important layers: hardware abstraction, operating system, and application. This layered concept provides convenient usage to efficiently use each resource and provide the modularity of each software components.

The routines to control individual hardware devices, such as manipulations of registers in the MCU, RF modem control, sensor and actuator control, are abstracted in hardware abstraction layer. Therefore, some changes in hardware would not affect upper layer software.

3.1 ELOS: Embedded Lightweight Operating System

In order to efficiently manage the limited resources and concurrently generated tasks, we developed an event-driven operating system called the Embedded Lightweight Operating System (ELOS) combined with a generic multi hop ad hoc routing protocol. In this subsection, some details of the presented ELOS system are presented.

First of all, it is considered that scheduling policy of an operating system is one of the important parameters determining overall system performance, since multiple tasks by various flows are processed according to the adopted scheduling policy. Therefore, in order to perform an efficient task scheduling satisfying requirements of real-time ubiquitous systems, a scheduler, which is a conditional preemptive FCFS scheduling method with a guaranteed time slot, is newly designed. The scheduler is adopted in our ELOS system. In the proposed scheduling method, tasks by general events are processed similarly to general FCFS, but the executions can be preempted by registered time-critical tasks. The time-critical task is also guaranteed

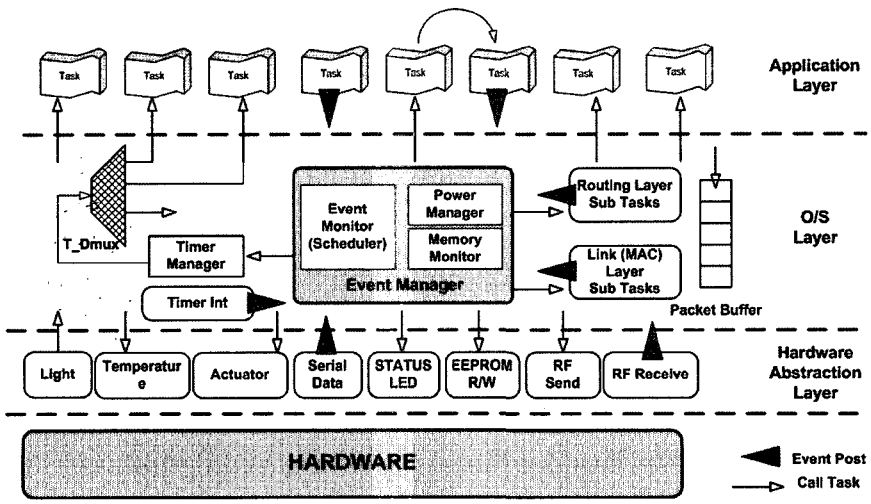


Figure 1 The proposed real-time software architecture for ubiquitous networked systems

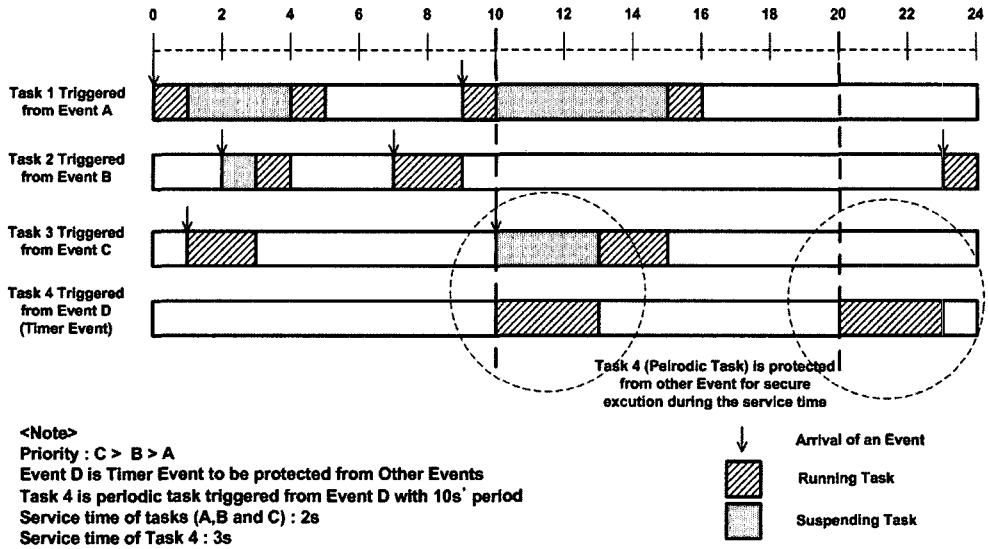


Figure 2 Operation of scheduling method used in ELOS

during its execution time without any interruption in deadline. Figure 2 shows the detailed operation of the proposed scheduling method used in the ELOS. In section V, it is shown that the proposed scheduling method outperforms traditional scheduling methods, such as FCFS, LCFS, Round-Robin, and Priority Scheduling, from the several viewpoints.

The ELOS performs event-driven task management. This event-driven method is suitable for application specific characteristics. The ELOS supports two kinds of event: sporadic event and periodic event. The former is an event that is not real-time and triggered from hardware or software, and the latter is a time-critical event. Each event is classified when they are registered. Therefore, the scheduler can protect time-critical tasks from other sporadic tasks.

A complete ELOS system consists of an *event manager* and set of *application components*. Event manager includes a *scheduler, power manager*, and *memory monitor*. An application component is composed of an *event list, event handler*, and a bundle of related *tasks*. Each application component registers the corresponding event in an event list and declares tasks to be called by the event.

An event is used as a signal to call a corresponding task in multiple data flows. In order to achieve high-level event management, the ELOS uses an event list. This event list plays an important role in buffering events, whereby the ELOS efficiently performs conditional preemptive FCFS scheduling with guaranteed time slots by reconfiguring the hardware and software events. This feature in the ELOS is distinguished from other event-driven operating systems in which event generations only depend on hardware interrupts. This also provides more programmable, flexible event management to application-level program.

A task creates a method to perform various arbitrary jobs. A task consists of one or more functions, and most tasks are triggered from corresponding events or other tasks. These tasks cannot preempt events or other tasks. Only registered tasks can be scheduled and processed by posting events.

The power manager performs power management of each hardware component and MCU by controlling the power mode. This power manager operates with a programmable PM value similar to a watchdog timer. The value is refreshed whenever events are triggered and decremented during an idle event. When the value reaches zero, the manager changes the state into idle. The idle state is recovered from interrupts. However, more detailed and aggressive power management is left

to the application.

The memory manager protects a memory from several violations during execution time, since burst traffic can make stack overflow in the code or data memory. In addition, the memory manager manages an internal EEPROM. The memory is used for backup of current important contents when the memory overflow or system faults occur..

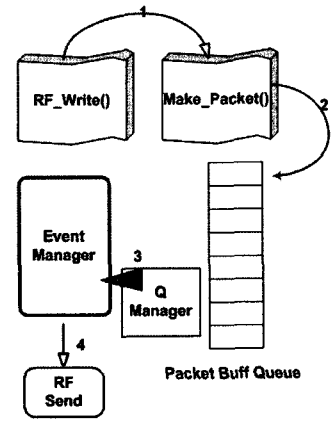
3.2 Communications in ELOS

The communication is one of the most important software components for a ubiquitous networked system, since most tasks are related to communication. The communication software in ELOS, which is based on the layered concept, in particular, is designed for short-range multi hop ad hoc networking.

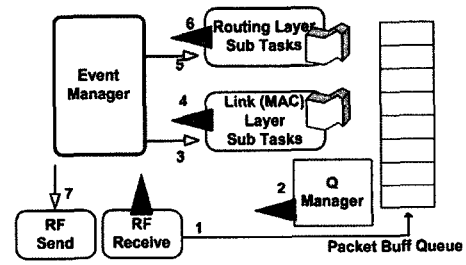
In the ELOS, all hardware control codes are abstracted into hardware abstraction layer so that each layer can be designed and operated without concern about the kind of RF modem. In addition, since the ELOS provides event posting function in software (tasks) as well as hardware interrupts, the event can be used as an asynchronous signal to communicate between each layer. This maximizes the independency between each layer.

In a resource limited real-time ubiquitous system, packet and queue handling becomes a primary concern. The ELOS uses a *packet buffer* for efficient packet handling and it is managed by a queue. The use of the packet buffer has some advantages. First, since higher layer blocks of data are encapsulated by lower layers, it is useful to design independent protocols for each layer. Also, the use of a packet buffer makes it possible to maximize memory utilization by minimizing duplicate memory copy. The ELOS provides basically a MAC header, Routing header, and Application layer header. Each field in each header can be easily modified, removed and added.

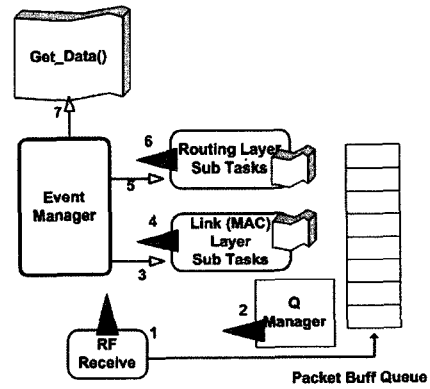
In addition, the ELOS is able to operate with various MAC protocols, which include CSMA/CA and TDMA. Moreover, the ELOS provides several routing protocols modified for more efficient routing on the basis of traditional routing protocols, a flooding, tree-based routing protocol, and mesh supported routing protocol.



(a) Source Node



(b) Intermediate Note (Router)



(c) Destination Node

Figure 3 An example of event-driven packet process for multi hop routing

Data delivery in the ELOS is achieved by Packet buffer, Queue manager, and MAC and Routing sub tasks on the basis of the Event manager. Figure 3 illustrates an example of event-driven communication process for multi hop routing, in source node (a), intermediate node (b), and destination node (c), respectively.

First, source node generates data when one of the following conditions is satisfied: periodic sensing data delivery, query flooding, and detection of over-threshold sensing event or active tag. Data Generated is stored in packet buffer queue as a packet structure by Make_packet task. Queue manager, which monitors periodically the status in queue, posts a read_queue event if new data occurs. The event manager schedules and executes the RF_send task. And then the packet is sent to FIFO of RF modem.

For the reception process, after completing error check, RF data obtained from RF modem is stored in the Packet buffer queue as a packet structure by the RF_Receive task. On noticing that the new packet arrives, the queue manager posts the event. The event manager executes MAC layer subtasks according to the event, and then the MAC subtask posts new event that notices that there is a packet to be processed at the upper layer (routing). Event manager executes Routing layer subtasks again, and if the packet's final destination address is matched with its own address as shown in (c) of Fig.3, the data is obtained from packet buffer queue by performing Get_data task. Otherwise, as shown in (b) of Fig.3, for re-forwarding, the packet is delivered to RF modem through the RF_send task. In this process, asynchronous events between each layer enhance the independency between each layer and provide more effective usage for packet handling.

In order to adapt to chaotic conditions such as intermittent failure and node destruction, the ELOS has the ability of self-organization based on the MAC and routing protocols. However, such sites, where well planned networks are required but exposed by numerable interferences, require more precise, reliable configuration methods than self-organization. In practical distributed ad hoc network implementation, several problems result from environmental variations, including asymmetric link, and communication interferences. These can bring about great confusion to the entire network. Moreover, these problems are hardly recovered using just self-organization because of its fundamental limitations. Therefore, in order to overcome such a limitation of self-organization, semi-auto configuration function is supplemented in the ELOS. This semi-auto configuration function is performed with a special node called a wireless configuration agent node. The agent has the ability to diagnose the status of neighboring problematic nodes and to force to change the routing path and other configuration information as shown in fig. 4. Therefore, it is possible that various topologies that a user wants are established without concern about physical deployment of nodes, guaranteeing creation of a well planned network.

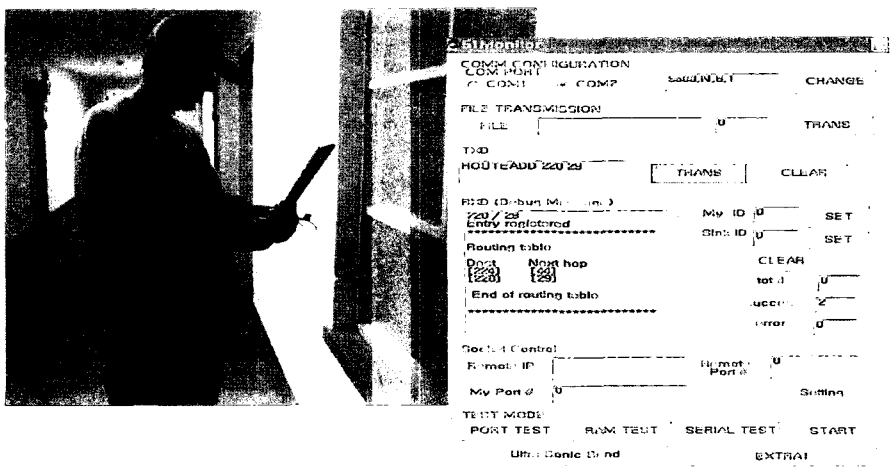


Figure 4 Semi-auto configuration process

4. Hardware Prototype

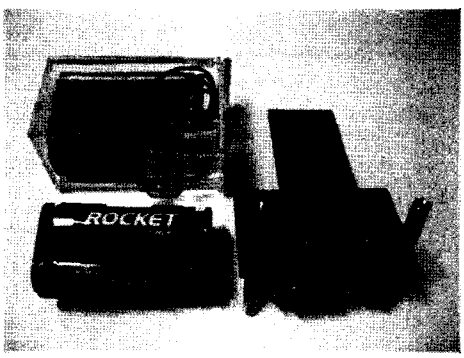
Although the proposed ELOS system is designed not depending on only a specific hardware platform, in order to implement and verify the proposed real-time ubiquitous networked system software, a prototype, with a very compact size of a square inch, powered from two 1.5V AA, is developed. As shown in Fig. 5, the hardware prototype is composed of four individual boards, Main board, RF boards, I/O boards and Interface board for interface with a PC or PDA.

The main board plays the most important role of driving the RF board and I/O board. An ATMEL AT89c51ED2 MCU is used as a main MCU with a 12MHz main clock. It includes 64kbytes of internal flash memory for code, and 2048 bytes of internal RAM. Power mode of MCU can be controlled by software with three power mode levels: ACTIVE, IDLE, and POWER DOWN. For efficient I/O connection, the prototype uses address decoded I/O mapping method. This has the advantage of being more flexible, scalable I/O connections compared to direct use of general purpose I/O ports.

The RF board is composed of a RF modem for short-range wireless communication and additional RF circuits, such as an impedance matching circuit and antenna. First of all, it is noticeable that the prototype supports four different kinds of modem: RF102 (RFwaves), nRF2401 (Nordic), CC2400 (Chipcon), and CC2420 (Chipcon, IEEE802.15.4 compatible). In order to interface individual RF modem, a common interface to connect between the RF board and Main board is devised. This common interface consists of a 3-wired Serial Peripheral Interface (SPI) and 4 general I/O ports, providing common usage for individual RF modems to the software developer.

To support various sensor and actuator applications, the I/O board is separated from the Main board. The I/O board includes basically a temperature sensor, a CDS cell, ultrasonic sensors, and a relay circuit to control external illumination or another actual system. The board can also accommodate additional I/O devices.

In general, most of the ubiquitous systems are battery powered. However, some specific nodes need to be connected to high level computing devices, such as a PC or PDA. For such nodes, the Interface board provides two kinds of communication interfaces, RS-232C and USB, to communicate between PC and an embedded system and powered by an AC adaptor. The interface board is also used as an interface to load a program or to perform serial debugging



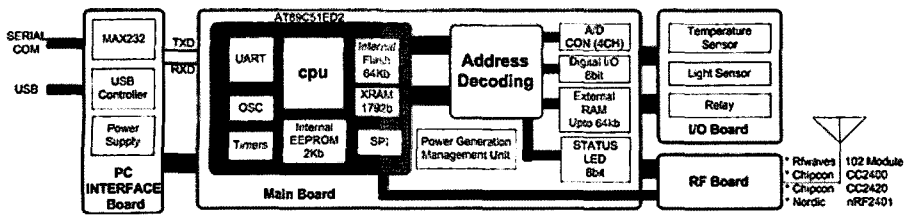


Figure 5 Hardware prototype in which the ELOS system is implemented

5. System Evaluations

In this section, the performance of the ELOS system, which is implemented on the developed hardware platform, is evaluated. The system is first evaluated from the viewpoint of a system level, and then, network level evaluation on the developed ubiquitous network test-bed is conducted.

5.1 System level evaluation

5.1.1 Small size and low-power consumption

Figure 6 shows the breakdown of the code and data for each the component in our complete system. It is remarkable that the code size of the complete ELOS, including several multi-hop ad hoc routing protocols, two types of MAC protocols, and self and semi-auto configuration software, is approximately 10kbytes. When considering total prototype flash memory size of 64kbytes, our system consumes just 15 percent of available memory. Note that data memory requires just 560 bytes of data memory space.

Table 1 shows the power consumption for each component in the hardware prototype. When using two 1.5V batteries with 1250mAh, the lifetime of the prototype can last for one and half years in fully idle mode, and for a week in fully active state. Therefore, practical lifetime can last for a couple of months to a year according to software power management.

5.1.2 Efficiency of Real-time Scheduler

Table 1 Power consumption in our hardware prototype

		Active	Idle	Power Down
MCU		7.9mA	6.6mA	75μA
Decoder Logic		20μA	-	-
A/D Con		1.5mA	-	1μA
Temperature Sensor		60μA	-	-
RFM	CC2420	Tx: 17.4mA Rx: 19.7mA	426μA	20μA
	CC2401	Tx: 19mA Rx: 23mA	1.2mA	1.5μA
	nRF2401	Tx: 13mA Rx: 25mA	12μA	400nA
	RF102	Tx: 21mA Rx: 38mA	-	26μA

component	Code	Data
Hardware	1018	19
Initial Process	441	12
Timers	285	19
Resource Manager (Scheduler)	270	22
Packet & Queue Management	2339	243
PHY	1609	42
MAC	1323	49
Routing	1711	79
Configuration	736	18
App	786	56
Total	10498	559

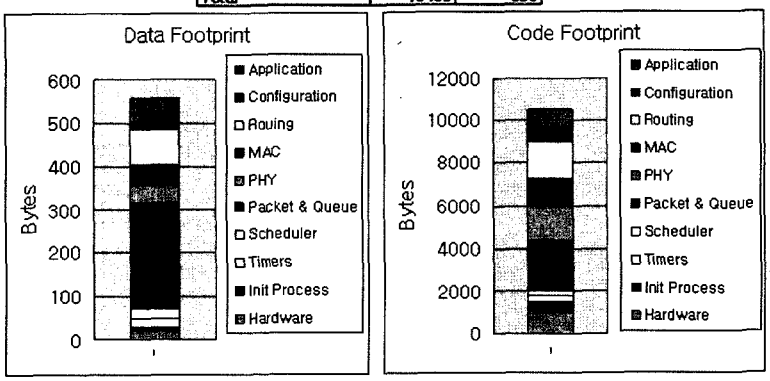
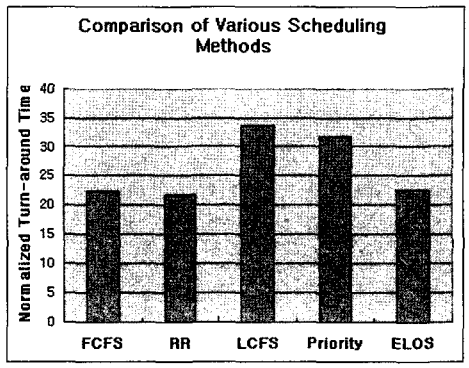


Figure 6 Breakdown and footprint of code and data for our completion system

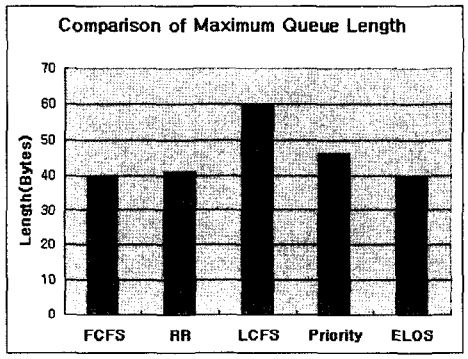
As presented in previous section, scheduling policy of an operating system is one of the most important parameters determining overall system performance, since multiple tasks by various flows are processed according to the adopted scheduling policy. Efficient real-time system should have the ability to accommodate multiple concurrency-intensive data flows and to process real time tasks as well as efficient management of limited resources. In section Ⅲ, we have already introduced a conditional preemptive FCFS scheduling with guaranteed time slots, which is adopted in the ELOS.

There are several traditional scheduling methods, such as, first-come-first-served (FCFS), last-come-first-served (LCFS), round-robin known as time-slicing, and priority queuing. The performance of these scheduling policies can be different according to the application environments in which they are used. We have examined the following three metrics for real-time scheduler: the first is the *normalized turnaround time*, which represents relative delay experienced in the system by a task. The second is the *maximum queue length*, which shows how much memory is required for scheduling tasks. The final measurement is the *guaranteed rates of time-critical tasks*, which represent how many time-critical tasks is completed without any interruption. All schedulers are tested on the developed prototype. It is assumed that the size of all tasks is very small, and the system generates a burst of concurrency-intensive events, with one time-critical task.

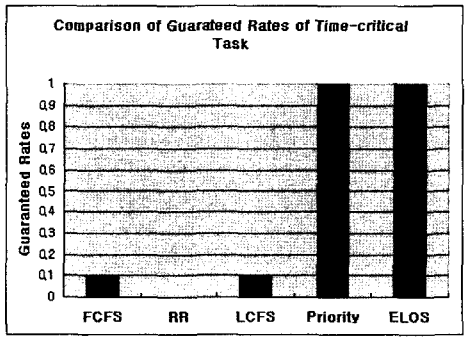
Figure 7 presents the results of observation of each experiment. Note that FCFS and round-robin scheduling methods outperform other scheduling methods from the point of the normalized turnaround time and maximum queue length, respectively. It is also shown that priority scheduling is superior to others in terms of guaranteeing tasks. However, round-robin scheduling is less recommended for real-time ubiquitous systems, because of intermittent application traffic characteristics in long run behavior. In addition, LCFS scheduling can bring about the potential data-race problem due to not mutual exclusive updates to shared memory. Besides,



(a) Comparison of Normalized Turnaround Time



(b) Comparison of Maximum Queue Length



(c) Comparison of Guaranteed timecritical tasks

Figure 7 Comparison of performance and guaranteed task of the several scheduling methods

frequent context-switching in small tasks increases the redundant switching overhead. Note that the proposed scheduling method, which is used in ELOS, shows better performance than others from the several viewpoints as shown in Fig. 7.

In addition to real-time scheduling ability, as shown in Fig. 6, the ELOS executes the resource

manager with a code size of just 270 bytes and data memory size of just 22 bytes, including the scheduler, power manager, and memory monitor.

5.2 Network level evaluation

To implement and verify the proposed system, a ubiquitous network test-bed is developed in our building. As shown in Fig. 8, the test-bed consists of 10 embedded nodes, which are deployed on the corridor, and 11 nodes, which are in the service room connected with the corridor, including 4 illumination sensor nodes, 4 temperature sensor nodes, 2 actuator nodes(one is to control illumination and another is for an air cooling system), and one sink node. Also, for user identification, a user's PDA is equipped with an active tag, which is an embedded node periodically broadcasting its ID. As shown in fig. 8, the nodes in the service room form a star topology and the nodes on the corridor construct a multi hop ad hoc network. A sink node is connected to the embedded gateway [13] to monitor and control the network though the Internet. The Internet is also used to provide the organizational map through local WLAN.

5.2.1 Real-time processing ability with respect to various traffics

The final goal of ubiquitous networks is to reliably perform distributed data aggregation to sink nodes. We first examined the successful aggregation rate of data acquired from multi-hop nodes in the test-bed. As shown in Fig. 9(a), it is noticeable that our system performs reliably data aggregation through the multi hop ad hoc networking, maintaining greater than 90% success rates.

In order to evaluate the real-time processing ability, complete response rates in various traffics are observed. First, each node responds to the sink node periodically. At that time, the PDA user (equipped with active tag) moves toward the sink node, whereby static nodes on the corridor or service room come to identify the active tag. Nodes

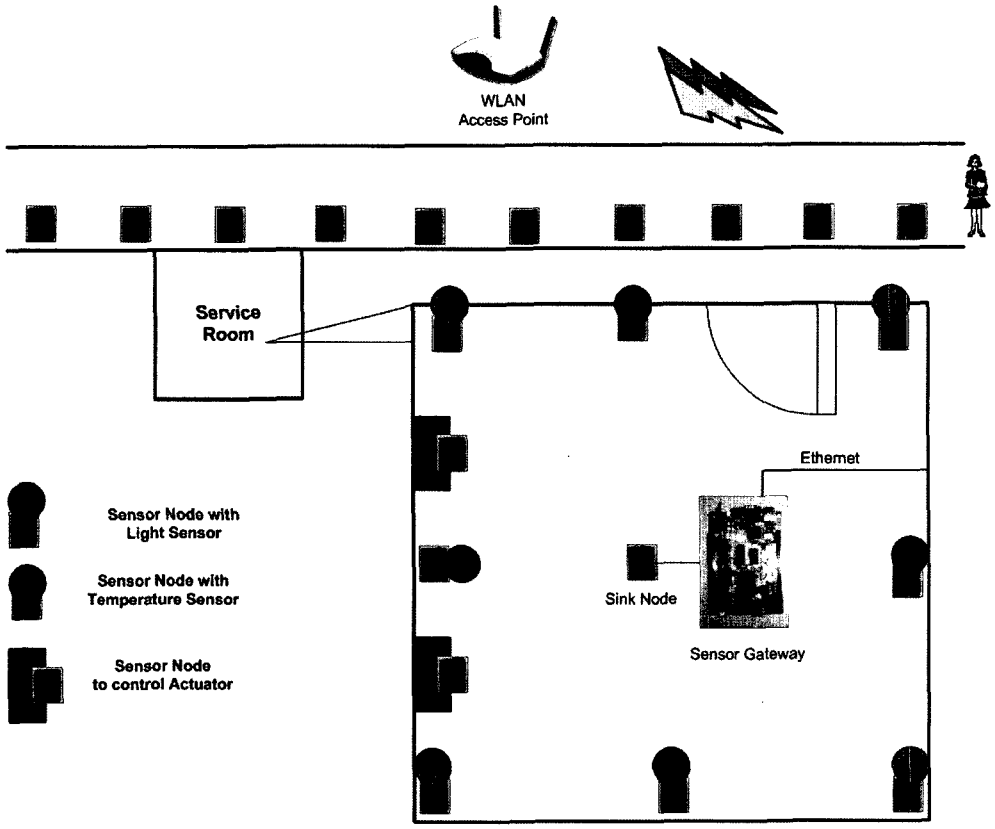
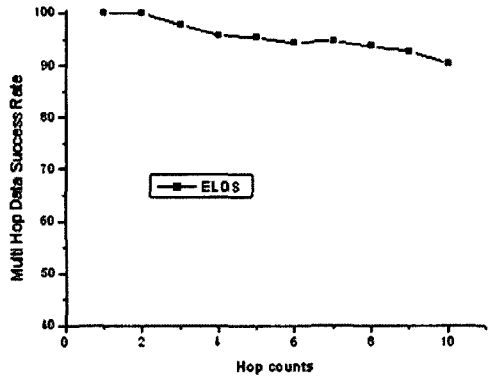
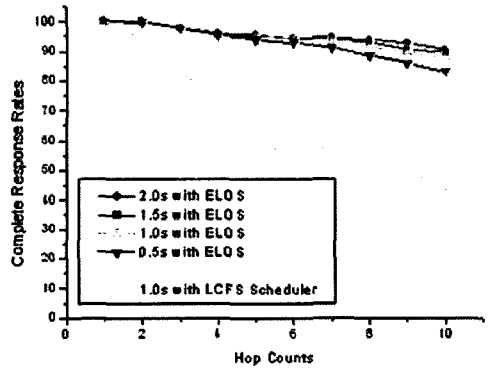


Figure 8 Real-time ubiquitous network test-bed environment



(a)



(b)

Figure 9 Multi hop data success rate and complete response rates

which sense the active tag send its location information to sink node, immediately. Eventually, this experiment constructs two different communication events concurrently: the first is a periodic event and the second is a sporadic event from the movement of the active tag. Under this situation, the ELOS and LCFS, which is adopted in TinyOS, are tested, respectively. In the simulation, the query period was varied from 0,5s to 2.0sec. As shown in Fig. 9(b), it is presented that the ELOS is superior to LCFS scheduling in terms of complete response rate with respect to varying traffic. It can be said that real-time processing ability of the ELOS presents considerably reliable performance in diverse traffic environments where the real-time tasks coexist with non-time critical tasks.

6. Conclusion and Future Work

We presented a network-oriented lightweight real-time system, which is composed of an event-driven operating system called the Embedded Lightweight Operating System (ELOS) and a generic multi hop ad hoc routing protocol suite. In the ELOS, a conditional preemptive FCFS scheduling method with a guaranteed time slot is designed for efficient real-time processing. For more elaborate configurations, we reinforce fault tolerance by supplementing semi-auto configuration using wireless agent nodes. The developed hardware platform is also introduced, which is a scalable prototype constructed using off-the-shelf components. In addition, in order to evaluate the performance of the proposed system, we developed a ubiquitous network test-bed on which several experiments with respect to various environments are conducted. The results proved that the ELOS is considerably favorable for tiny ubiquitous networked systems with real-time constraints.

In the future, for more complete large-scale real-time ubiquitous networks, it is important that the ubiquitous networked nodes should be scaled down to a microscopic size, utilizing advances in MEMS and CMOS logic technologies. In addition, employing a renewable energy source such as solar energy will solve the energy problem. In addition, real-time system software is expected to be capable of more autonomous, powerful self-testing, self-healing, and self-updating.

References

[1] G. Asada, T. Dong, F. Lin, G. Pottie, W. Kaiser, and H. Marcy, "Wireless integrated network sensors: Low power systems on a chip," in *European Solid State Circuits Conference.* October 1998.

[2] J. Rabaey, J. Ammer, T. Karalar, S. Li, B. Otis, M. Sheets, T. Tuan, "PicoRadios for Wireless Sensor Networks: The Next Challenge in Ultra-Low-Power Design," *Proceedings of the International Solid-State Circuits Conference*, San Francisco, CA, February 3-7, 2002.

[3] Rex Min, Manish Bhardwaj, Seong-Hwan Cho, Amit Sinha, Eugene Shih, Alice Wang, and Anantha Chandrakasan, "Low-Power Wireless Sensor Networks," *VLSI Design 2001*, January 2001.

[4] Sung Park, Ivo Locher, Mani Srivastava, "Design of a Wearable Sensor Badge for Smart Kinder-

garten," *6th International Symposium on Wearable Computers (ISWC2002)*, Seattle, WA, October 7-10, 2002.
[5] A. Savvides and M. B. Srivastava, "A Distributed Computation Platform for Wireless Embedded Sensing," in the *Proceedings of International Conference on Computer Design 2002*, Freiburg, Germany. 2002.
[6] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, Kristofer Pister, "System Architecture Directions for Networked Sensors," *ASPLOS 2000*, 2000.
[7] PalmOS Software 3.5 Overview. http://www.palm.com/devzone/docs/palmos35.html.
[8] pSOSystem Datasheet. http://www.windriver.com/products/html/psosystem_ds.html.
[9] VxWorks 5.4 Datasheet. http://www.windriver.com/products/html/vxwks54_ds.html.
[10] Microsoft Corp. Microsoft Windows CE. http://www.microsoft.com/windowsce/embedded/
[11] M. Chiodo. Synthesis of software programs for embedded control applications, 1995.
[12] Barry Kauler. CREEM Concurrent Real-time Embedded Executive for Microcontrollers. http://www.goofee.com/creem.htm.
[13] Kwang-il Hwang, Jeongsik In, Nokyoung Park, and Doo-seop Eom, "A Design and Implementation of Wireless Sensor Gateway for Efficient Querying and Managing through World Wide Web," *IEEE Transactions on Consumer Electronics*, vol. 49, Nov. 2003, pp. 1090-1097.
[14] Perkins, *Ad Hoc Networking*, Addison Wesley, 2001.
[15] Ian F. Akylidiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci, "Survey on Sensor Networks," *IEEE Communication Magazine*, 2002.
[16] IEEE P802.15.4/D18, Draft Standard: Low Rate Wireless Personal Area Networks, Feb.2004.

황 광 일

2002년 홍익대학교 전자전기전파공학부 학사. 2004년 고려대학교 전자컴퓨터공학과 석사. 2007년 고려대학교 전자컴퓨터공학과 박사. 현재 시립인천전문대학 컴퓨터제어과 전임강사. 관심분야는 AD HOC networks, RFID systems, Sensor networks, Ubiquitous networks

엄 두 섭

1987년 고려대학교 전자공학과 졸업. 1989년 고려대학교 전자공학과 석사. 1999년 일본 오사카대학 정보컴퓨터과학과 박사 1996년~1999년 ETRI 선임연구원. 1999년~2000년 원광대학교 전임강사. 현재 고려대학교 전자공학과 부교수. 관심분야는 AD HOC networks, RFID systems, Sensor networks, Ubiquitous networks, Internet QoS