Special issue on International Workshop on Networked Sensing System

Trans. of the Society of Instrument and Control Engineers
Vol.E-S-1, No.1, 74/84 (2005)

PAVENET: A Hardware and Software Framework for Wireless Sensor Networks

Shunsuke Saruwatari *, Takuya Kashima **, Masateru Minami ***,
Hiroyuki Morikawa * and Tomonori Aoyama **

Wireless sensor networks are attracting attention as a neural system in the coming ubiquitous computing environment. Building a realistic architecture requires a testbed that meets various demands. Therefore, we have designed and implemented a wireless sensor network testbed called PAVENET. PAVENET consists of a hardware module U^3, U^3 SDK, which is a development kit for U^3 software, and basenode software, which supports the development of wireless sensor network technology. PAVENET has four characteristic features: hardware level modularization, dual-CPU architecture, hard real-time transaction support, and network layering APIs. These features help us to develop not only application functions, but also wireless communication functions. This paper describes implementation details and a number of implemented applications, such as ANtennary THings, a traffic line detection system, and media access control (MAC) performance evaluation system.

Key Words: Wireless Sensor Networks, Embedded Systems, Ubiquitous Computing, Middleware

1. Introduction

Sensor networks can relay real-world information into the virtual world that is constructed by computer networks. We can create various services using these sensor networks. As a primitive application of sensor networks, a simple data gathering service has attracted a great deal of attention from researchers. In order to realize these primitive applications, a number of wireless communication technologies, such as routing protocols and MAC (media access control) schemes, have been proposed [1)~3)].

The dawn of sensor network technology is coming to an end, and new applications of sensor networks are desired. Thus, a sensor network testbed is necessary in order to develop innovative applications using sensor network technology. The testbed would also have to equip functions for the development of new wireless communication protocols that support the applications.

There are several testbeds for wireless sensor networks, including MICA Mote [4)] and Smart-Its [5)]. However, using these testbeds for next-generation sensor network services is difficult because these testbeds are designed as single-CPU architecture in order to construct simple services, such as data gathering. The services, which can be constructed by single-CPU architecture, are constrained because software tasks for wireless communications consume a great deal of computing resources. In order to realize these innovative applications, the next-generation sensor network testbed should not limit application development.

Based on these considerations, we design and implement a hardware and software framework for wireless sensor networks, called "PAVENET", that supports the construction of various applications. PAVENET includes U^3, which is a dual-CPU wireless sensor node, U^3 SDK, which is a software development kit for U^3, and basenode software, which supports application development and operates sensor networks.

PAVENET has the following characteristic features;

- hardware level modularization
- dual-CPU architecture
- hard real-time transaction support
- network layering APIs

PAVENET divided the functionality of the sensor node into four physically separated modules: a power control module, a system software module, a communication module, and a device module. This hardware level modularization enables us to independently replace each function, such as power control, wireless communication devices, CPU, and sensors. On the other hand, conventional sensor network nodes, such as MICA Mote [4)], only allow replacement of the sensor board.

* School of Frontier Sciences, The University of Tokyo
** School of Information Science and Technology, The University of Tokyo
*** Department of Electronic Engineering, Shibaura Institute of Technology

PAVENET has dual-CPU architecture, even though most existing testbeds for sensor networks are built using single-CPU architecture. This dual-CPU architecture provides a number of advantages, including the ability to develop applications and communication protocols easily, the ability to evaluate communication protocols easily, and assistance when considering future developments on the hardware level.

Hard real-time transaction support for application software is realized by dual-CPU architecture and a simple priority based task scheduler. Most existing testbeds are constructed based on single-CPU architecture. However, MAC transactions in wireless communication may require real-time transaction support, and the transaction tasks occupy computing resources. Accordingly, application software is highly constrained by the MAC transaction task. PAVENET adopts dual-CPU architecture, expanding the capabilities of applications that can be developed.

Network layering is needed in order to more easily develop wireless communication functions. In order to realize the network layering, we separated the network functions into several parts and define simple interfaces between them. Therefore, PAVENET can enable us to develop functions more easily and simply than existing software frameworks, such as nesC [6] and MANTIS [7].

In order to evaluate PAVENET as a testbed, we developed ANTH (ANtennary THings), which is a real-space programming framework [8], and a traffic line detection system. Furthermore, we developed the 802.11-like CSMA MAC protocol and a number of measurement tools using PAVENET and conducted performance measurement of the MAC protocol.

The remainder of this paper is organized as follows. Section 2 describes the characteristics of the wireless sensor network, and extracts requirements in order to develop a testbed. Section 3 describes in detail the U^3, U^3 SDK, and basenode software, which are implemented based on extracted requirements in Section 2. Section 4 evaluates the proposed framework through a number of applications and an experiment. Section 5 describes a comparison between our testbed and the other testbeds as related work, and Section 6 concludes the present paper.

2. Design

In this section, we clarify the characteristics of wireless sensor networks and extract the requirements for the wireless sensor network testbed.

2. 1 Characteristics of wireless sensor networks

Wireless sensor networks have two characteristics: miniaturization and low power consumption. In regard to wireless sensor networks, a smaller sensor node means broader application, because installation locations, such as walls or ceilings, of wireless sensor nodes vary widely. For example, the Smart Dust project assumes extremely small sensor nodes to be floating in the air [9]. Moreover, because some nodes, such as mobile sensor nodes, are battery-powered, wireless sensor networks require a low-power-consumption architecture that can run continuously over several years. If we assume indoor use of wireless sensor networks, then we can also assume the existence of power-supplied nodes. However, we cannot ignore power consumption considerations because a slight increase in the power consumption of each node results in a huge increase in the power consumption of the entire wireless sensor network system. This is because wireless sensor networks have from several hundreds to several thousands of nodes.

However, limiting the creativity of application development by excessively prioritizing miniaturization and low power consumption is undesirable. Currently, the most important objective is to create novel sensor network applications. We should work toward the realization of miniaturization and low power consumption only after extracting elemental functions for applications.

2. 2 Requirements

PAVENET is an attempt to construct a hardware/software framework that can be used not only to create attractive sensor network applications, but also to extract basic functions that support it.

Toward this purpose, hardware/software level modularization and consideration of which function will be realized by hardware/software transaction in the future are important. A trade-off exists between modularization and miniaturization. If we would like to realize a sensor node of very small size and low price, all functions should be implemented on one LSI chip. However, implementing all functions on one LSI chip reduces flexibility, and improving individual functions or adding new functions is difficult. Since sensor network technologies are not mature, PAVENET gives greater priority to modularization than to miniaturization.

In order to realize modularization, we adopt not only software-level modularization, but also physical-level modularization. There is a wide range of potential applications and protocols for sensor networks. For environmental monitoring applications, a generic sensor interface would be useful for wireless nodes, so that the sensors could be easily replaced. Solar panels and rechargeable

batteries would also be advantageous for practical applications. Moreover, the user may wish to select the appropriate CPU and wireless communication devices according to power consumption or processing speed requirements, or according to instructions that will be given to the sensor. In order to meet these requirements, the hardware components of the sensor node should be constructed as an ensemble of independent modules that can be easily replaced. However, conventional sensor network nodes, such as MICA Mote [4], only allow replacement of the sensor board. Other components, such as the CPU, the wireless communication module or the battery module, are not replaceable.

To this end, we divided the functionality of the sensor node into four physically separated modules: a power control module, a system software module, a communication module, and a device module.

In addition, PAVENET adopts dual-CPU architecture because sensor network functions can be classified into two parts: wireless communication functions and functions for applications. This dual-CPU architecture provides the following advantages.

First, this architecture enables applications and communication protocols to be developed easily. Generally, the application developer and the communication protocol developer are not the same. Therefore, being able to disregard overhead between application tasks and communication tasks is important in order to facilitate software development. In addition, this enables software to easily support hard real-time processing because the wireless communication functions and application functions can independently appropriate computing resources.

Next, this allows us to easily evaluate communication protocols such as MAC protocols. Since communication transactions and application transactions are separate on the hardware level, we can independently evaluate the characteristics of the communication protocol, such as power consumption. The power consumption is not proportional to the number of CPUs, but rather to the number of tasks, because CPUs that are in the idle state do not consume much electric power. Energy efficiency can be categorized into two types. The first is that which is achieved through the improvement of algorithms such as MAC or routing schemes. In order to realize this kind of energy efficiency, several researchers have proposed a number of energy efficient communication protocols [1)~3)]. The second type of energy efficiency is that which is achieved through the improvement of the device structure or materials. Implementing the entire system in LSI achieves miniaturization and low power consumption simultaneously. However, it also has the disadvantages of not being able to be modified and a lack of flexibility. Therefore, we focused on the first type of energy efficiency, which is achieved by improvement of the relevant algorithms, indicating the importance of the ease with which communication protocols can be evaluated.

Finally, in the future, it will be helpful to consider the classification of tasks into hardware transaction tasks and software transaction tasks. In order to realize low power consumption and miniaturization of wireless sensor nodes, a number of the functions of the wireless sensor node must be implemented on the hardware level. In particular, simple, steady, parallel, and high-speed software transactions consume an enormous amount of electric power, but transacting these functions in hardware is less power consuming than in software. Therefore, we assume that simple adhoc routing functions, MAC functions, and physical layer functions, such as error correction, will be implemented on the hardware layer.

In PAVENET, we realize not only physical level modularization, but also software level modularization. Since wireless communication protocols are a very important feature in wireless sensor networks, we define several APIs that provide layered abstracted communication structure (application, media access, and physical layer). As a consequence of this software level modularization, users can select several protocols according to application demands, such as power consumption, and latency.

3. Implementation

According to the design mentioned reported in Section 2, we implemented a hardware and software framework for wireless sensor networks called PAVENET, which provides support for the construction of various applications. PAVENET includes U^3, which is a dual-CPU wireless sensor node, U^3 SDK, which is a software development kit for U^3, and basenode software, which supports the development of applications and the operation of wireless sensor networks from PCs or PDAs.

3.1 U^3

U^3 realizes hardware level modularization and dual-CPU architecture.

PAVENET divides the functionality of the sensor node into four physically separate modules: a power control module, a system software module, a communication module, and a device module. This hardware level modularization enables us to independently replace each function, including power control, wireless communication de-

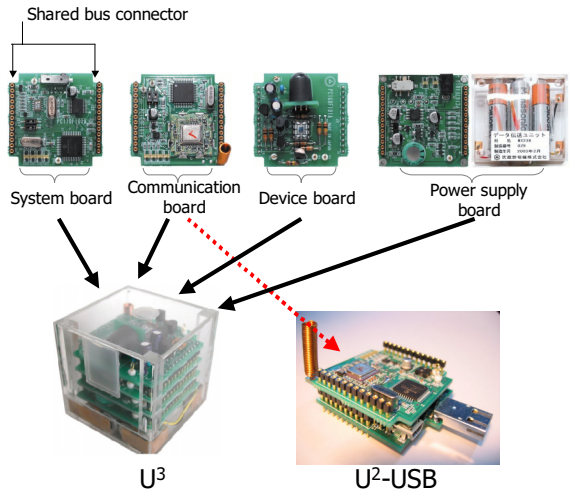


Fig. 1 Hardware organization

vices, CPU, and sensors. In contrast, conventional sensor network nodes, such as MICA Mote [4], only allow replacement of the sensor board.

In addition, PAVENET has dual-CPU architecture, although most existing testbeds for sensor networks are built using single-CPU architecture. One CPU is for application tasks, and the other is for communication tasks. This dual-CPU architecture provides the following advantages: easy development of applications and communication protocols, easy evaluation of communication protocols, and consideration of future hardware developments.

Figure 1 shows each of the four boards, along with U^3 and $U^2 - USB$, which are constructed using these four boards.

U^3 is a 50 mm × 50 mm × 50 mm wireless sensor box that contains four function boards: a power board, a system board, a communication board, and a device board. The boards are connected to each other through a 2.54 mm pitch shared bus connector.

U^2-USB, which consists of a communication board and an I^2C-USB conversion board, is a communication interface to control wireless sensor nodes from PCs or PDAs.

Each board is described below in detail.

The power supply board has three AAA 700 mAh nickel metal-hydride batteries and an external DC input for recharging the batteries. Furthermore, the power supply board supplies 3.0 V to the other boards, provides information regarding battery life, and outputs current to the shared bus connector. The system board can learn the remaining battery life as well as the energy consumption through the shared bus connector.

The communication board consists of an RF Monolithics 315 Mhz transceiver TR3001, a Microchip 8 bit MCU PIC18F452 that runs at 20 MHz, and a helical antenna. The PIC18F452 has 8 bit registers, a 1.5 Kbyte data memory, a 16 Kbyte program memory, and a 256 byte EEPROM. The PIC18F452 controls the TR3001, and processes the communication software of the U^3 SDK. The communication board provides I^2C interface to the shared bus connector.

The system board consists of IrDA and the same Microchip PIC18F452 as the communication board. The PIC18F452 runs at 10 MHz and processes the system software of the U^3 SDK.

The device board can equip various sensors or actuators, which are controlled by the system board. The device board provides interfaces to the shared bus connector. The interfaces include voltage, which relates information such as temperature, a port that relates 1 bit data, a port that controls the device, and an I^2C interface.

We have implemented several device boards, including a sensor board that has a motion sensor, a temperature sensor, and an illuminance sensor, a light board that has eight full-color LEDs, and a traffic line detection board.

3.2 U^3 SDK

U^3 SDK realizes hard real-time transaction support and network layering.

Hard real-time transaction support for application software is realized by dual-CPU architecture and a simple priority-based task scheduler. Most existing testbeds are constructed using single-CPU architecture. However, MAC transactions in wireless communication may require real-time transaction support, and these transaction tasks require computing resources. Accordingly, application software is highly constrained by the MAC transaction task. In contrast, PAVENET adopts dual-CPU architecture, thus expanding the capabilities of applications that

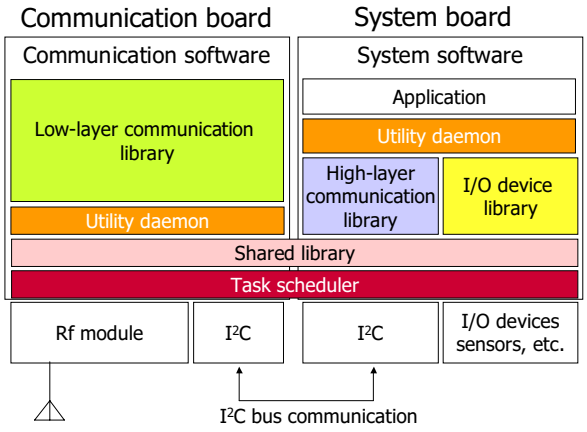


Fig. 2 U^3 SDK

may be developed.

Network layering is needed in order to easily develop wireless communication functions. In order to realize network layering, we separate network functions into several parts and define simple interfaces between these parts. Therefore, PAVENET can enable wireless communication functions to be developed more simply than existing software frameworks, such as nesC [6] and MANTIS [7].

U^3 SDK is a software development kit that supports system software and communication software, which are processed by a system board and a communication board, respectively. We use a HI-TECH Software PICC18 compiler for the development of U^3 SDK. U^3 SDK is designed with network layering and support in mind in order to realize various user demands.

Figure 2 shows the structure of U^3 SDK. Both the system software and the communication software have the same task scheduler and a shared library. The task scheduler has lightweight multithreading architecture, and supports hard real-time transaction. The hard real-time transaction can be simply implemented using two-level priority interruption of the PIC18 architecture. High-priority tasks that are bound to highest priority interruption can preempt low-priority tasks. The shared library consists of various APIs for task operations, such as add_task, trigger_task, and resume_task, and for debugging operations, such as exit_u3.

System software

The system software consists of a high-layer communication library, an I/O device library, and a utility daemon. Since the components are triggered by interruption and the transactions are designed to complete immediately, a large percentage of the CPUs can be maintained in the idle state, thus saving a great deal of electric power.

Figure 3 shows the network layering and the APIs in the high-layer communication library. The library transacts operations such as data aggregation and adhoc routing. Packets should be routed data centrally and application specifically in wireless sensor networks [10]. The application layer provides an interface between the network layer and peripheral devices such as sensors and IrDA. The network layer provides a framework for developing routing protocols. Users can expand functions, such as routing, by describing transaction in event handlers such as on_net_recv. The library also provides APIs, which enable us to gain independence from various protocols, such as set_net_opt or get_net_opt, that are used when setting the destination addresses or obtaining source addresses. Hence, we can experiment with replacing network protocols through trial and error. The control interface layer provides an interface between network functions and the utility daemon.

The I/O device library provides a structure that abstracts interfaces such as I^{2}C, UART, ADC, and IrDA, by open/read/write functions. Using the abstracted APIs of the I/O device library, we can create various portable software modules.

Figure 4 shows how utility daemons in the communication software and the system software work. The utility daemons support management and development of a wireless sensor node cooperating with a utility daemon of the communication software. The daemons are implemented using the I/O device library and the shared library. It receives various commands, such as setting a node address or modifying a new program via IrDA on the system board, and accesses network functions via the control interface layer.

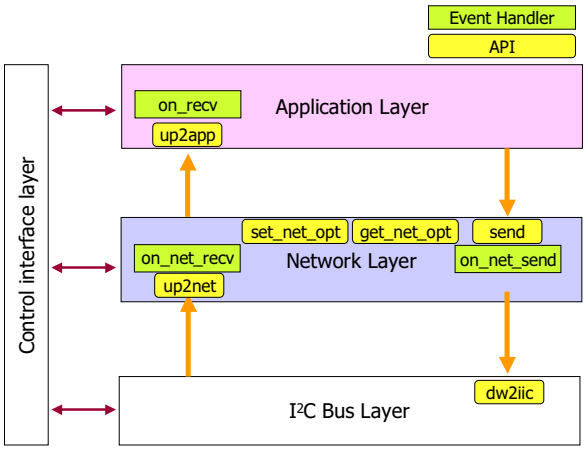


Fig. 3 High-layer communication library

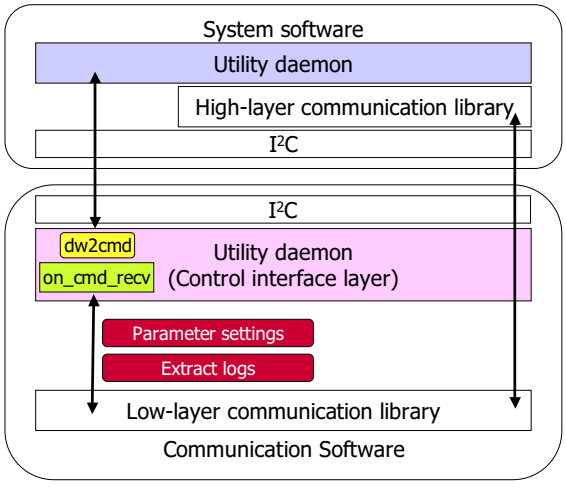


Fig. 4 Utility daemon

Communication software

The communication software consists of a low-layer communication library and a utility daemon.

Figure 5 shows the network layering and APIs in the low-layer communication library. The low-layer communication library consists of communication functions that will be implemented by hardware transaction in the future. In PAVENET, these functions are implemented by software considering prospective implementation in hardware, due to software flexibility.

A characteristic feature of the low-layer communication library is support for certain adhoc routing functions, such as source routing, and flooding, in the simple network layer. These simple adhoc routing functions can be realized in small memory by simple operations. Hence, these functions can be implemented by hardware transaction in the future. Transacting an adhoc routing function as a part of the communication software reduces the load of the system software and will accomplish prospective low power consumption in the future.

The utility daemon supports the management and development of a wireless sensor node, in cooperation with a utility daemon of the system software. The utility daemon provides functions, such as the setting of protocol parameters, the recording of communication logs, using the control interface layer.

3.3 Basenode software

The basenode software runs on PCs or PDAs to utilize wireless sensor networks and consists of a command line utility, a protocol translation gateway, and a basenode library. The command line utility can be used from the terminal of a PC via system board IrDA or U^2-USB. The utility includes pvnload, which loads a program to U^3, pvnget/pvnset, which can get/set parameters of U^3, and pvnping, which is used for confirming the existence of sensor nodes or for the measurement of packet delivery time. The protocol translation gateway works on PCs that equip a U^2-USB and allows users to access sensor networks through the Internet. The basenode library provides APIs that are arranged from functions of the command line utility and is used when the user develops an application for wireless sensor networks without command line utility.

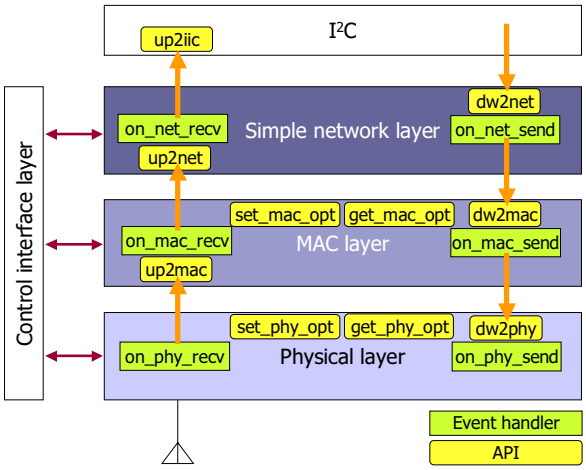


Fig. 5 Low-layer communication library

4. Evaluation

In this section, we describe a number of evaluations of PAVENET: a sample code, implementation of two applications using PAVENET, and the simple network performance evaluation of U^3. We verify the availability of PAVENET as a development environment and the experimental environment through these evaluations.

4.1 Sample code

In order to ensure the simplicity of PAVENET, we compare sample codes that are written using nesC [6] and U^3 SDK. nesC has been used to implement TinyOS [11]. nesC aims to realize a programming language for networked embedded systems that represent a new design space for application developers. The sample code makes LED blink every 1 second, which is a very simple operation.

In nesC, the user has to implement two different source codes, a configuration file and a module file [12]. An example of a configuration file is as follows:

```
configuration Blink{
}

implementation {
   components Main, BlinkM, SingleTimer, LedsC;
   Main.StdControl -> Blink.StdControl;
   Main.StdControl -> SingleTimer.Timer;
   BlinkM.Timer -> SingleTimer.Timer;
   BlinkM.Leds -> LedsC;
}
```

The following is an example of a module file:

```
module BlinkM {
   provides {
      interface StdControl;
   }
   uses {
      interface Timer;
      interface Leds;
   }
```

```
}

implementation {
   command result_t StdControl.init() {
      call Leds.init();
      return SUCCESS;
   }
   command result_t StdControl.start() {
      return call Timer.start(TIMER_REPEAT, 1000);
   }
   command result_t StdControl.stop() {
      return call Timer.stop();
   }
   event result_t Timer.fired() {
      call Leds.redToggle();
      return SUCCESS;
   }
}
```

In nesC, we have to not only learn a new language, but also to describe these redundant programs.

Next, sample code performs the same operation using the proposed framework.

```
void user_init(void)
{
    add_task(sample1, TASK_NO_PRIORITY);
    trigger_task(sample1);
}

void sample1(void)
{
    set_led0(~(get_led0()));
    wait_timer(1000);
}
```

PAVENET enables us to describe arbitrary operations more simply than nesC. Moreover, PAVANET adopts the C language, which is one of the most widely used languages in the world, hence many people can utilize PAVENET more easily than nesC.

4.2 ANtennary THings

We are developing a real-space programming framework called ANtennary Things (ANTH) using PAVENET. Through this implementation, we verified the effectiveness of the hardware level modularization of PAVENET by implementing a number of devices using the proposed framework.

In a ubiquitous computing environment, communication and computation functions are embedded in all of the object that surround us, which enables us to synthesize various services by combining these objects.

Many service coordination frameworks, such as UPnP [13], Jini [14) 15)], and Bluetooth [16)], have been proposed. These frameworks are useful for constructing conventional or static services because they are designed for configuring devices automatically or replacing cable networks with wireless networks. These technologies eliminate annoying entwining cables and complex device configurations. However, they do not provide a creative environment that enables new services to be constructed because they are designed while specifying existing services.

In view of this, we are developing a real-space programming framework called ANtennary THings (ANTH) for the ubiquitous computing environment. ANTH attempts to construct a programmable real-space that enables us to create desired services by ourselves. ANTH provides a chip that has three characteristic functions: a user interface function that controls the connection of one device to another, a communication function that constructs a network infrastructure for device cooperation, and a computation function, which drives devices and processes applications. The chip is referred to as an ANTH chip, and an object equipped with an ANTH chip is referred to as an ANTH object. We assume, in the future, that ANTH chips are embedded in all everyday objects, such as alarm clocks, lights, and walls. We can create various services by combining these objects. For example, a motion sensor event bound to an alarm clock bell realizes an instant security system, which notifies the user of an intruder by ringing the bell.

Figure 6 shows four implemented ANTH objects: a light, an alarm clock, a button, and a U^3, which is a wire-

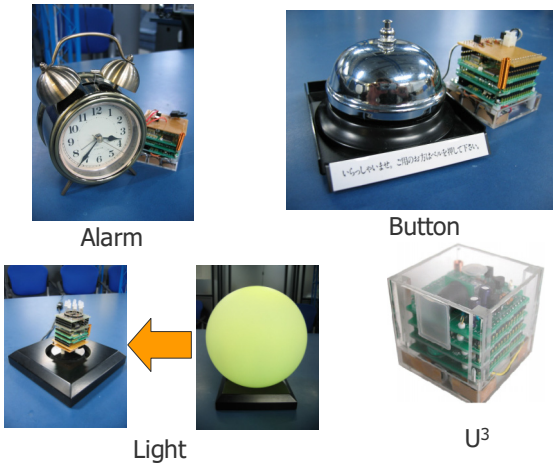


Fig. 6 Prototype implementation

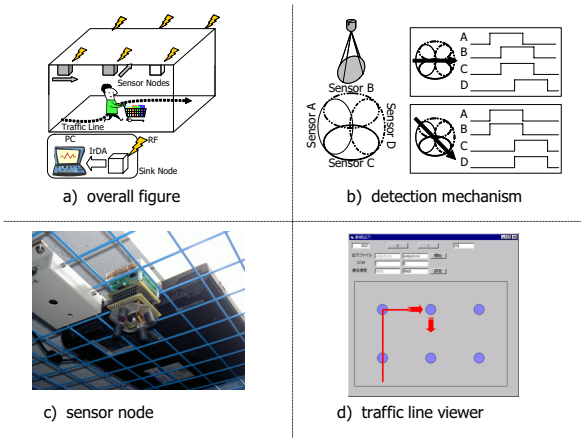


Fig. 7 Traffic line detection system

less sensor node. These objects are implemented using PAVENET, and we have tested some of the operations of ANTH using these objects [8]. The light has a full color LED board as a device board, and a domestic electric board as a power board. The alarm clock has an alarm clock board that controls an alarm bell as a device board, and a battery board as a power board. The light and the alarm clock have different device boards and power boards but have the same system board and communication board.

4.3 Traffic line detection system

We have implemented a traffic line detection system using PAVENET. Through this development, we verified the effectiveness of the dual-CPU architecture of PAVENET because calculating direction from acquired sensor data is a very large task.

Traffic line information is used for various applications, such as the placement of items in a supermarket, analysis of congestion conditions in an exhibition hall, and the placement of appliances in an office.

Figure 7-a shows an outline of the traffic line detection system. This system consists of multiple sensor nodes that have a direction detection sensor and a sink node that extracts direction information from the sensor nodes.

Figure 7-b shows the mechanism of the direction detection sensor. The direction detection sensor consists of four spot type motion sensors (Matsushita AMN13111). The motion sensor has an oval detection area and can detect infrared radiation generated by human body. The detection areas of the motion sensors partially overlap. The direction detection is realized by calculating the differential of reactions of them.

Figure 7-c shows a prototype system that is implemented using PAVENET. Sensor nodes and a sink node are implemented using U^3. We examined the quality of the direction detection sensor. We placed a sensor node on the ceiling of our laboratory. We then measured the successful detection ratio for detecting a person passing in a single direction under the sensor node. The detection successful ratio was 95%.

In the system board, there exists the very large task of calculating direction from acquired motion data. First, the task acquires binary data from four motion sensors every 10 msec. Next, the task removes noises from the series of binary data in time axis. Finally, the task extracts a direction based on the timing of the rising edges and falling edges of the four motion sensors and sends the direction data to a sink node. Although the above task could not be implemented using MICA and TinyOS [4), 6), 11)], the task could be implemented using the proposed framework because of the dual-CPU architecture.

Figure 7-d shows an example of the behavior of a traffic line viewer that was developed using Microsoft Visual Basic 6.0. The traffic line viewer receives direction information from the sink node via a protocol translation gateway of PAVENET and plots a traffic line.

4.4 MAC performance evaluation

In this measurement, we evaluate effectiveness of PAVENET as an experimental environment for wireless communication protocols.

To this end, we implemented an 802.11-like CSMA protocol as MAC, and a flooding routing protocol to U^3 using U^3 SDK. The flooding routing protocol broadcasts all received packets when the packet is received for the first time. In addition, we implemented a performance evaluation tool using the basenode software that works on a PC.

Figure 8 shows the measurement results for successful packet receive ratio according to the number of nodes and the contention window size. The measurement used one

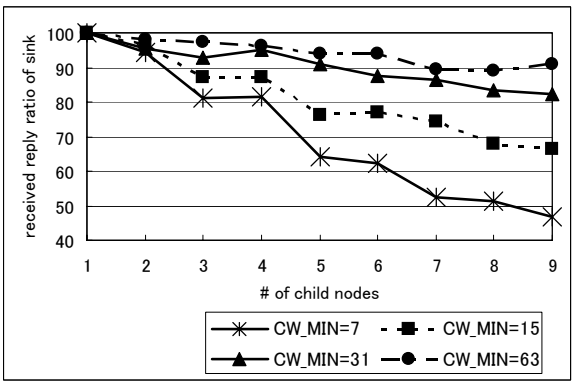


Fig. 8 Protocol experiments

sink and a number of sensor nodes, and all of the nodes can communicate with each other. First, the sink broadcasts a query to the sensor nodes. Next, the sensor nodes that received the query send a reply to the sink. The communication load of the sink is very high because all of the sensor nodes that received the query send a reply together and the routing protocol is flooding.

Figure 8 shows the measurement results. The ratio is nearly 100% when communication is peer-to-peer, but the ratio decreases notably when the contention window size decreases and the number of nodes increases.

Users can perform these measurements easily, because PAVENET provides a command line utility that enables us to change network parameters such as the contention window size of CSMA.

5. Related studies

Several testbeds for wireless sensor networks already exist.

There are a number of technologies that assume a single-CPU architecture.

The Particle Computer, which is part of the Smart-Its project [5], is developed at TecO, University of Karlsruhe [17]. The Particle Computer is a platform for rapid prototyping of ubiquitous computing environments for adhoc (sensor) networks, wearable computers, home automation, and ambient intelligence environments. Therefore, the Particle Computer attaches importance to miniaturization and the development of applications that are available for immediate use. The Particle Team [18] of the Smart-Its project is a very creative group, and they have developed many applications for the Particle Computer [19)~26)]. In order to realize these applications, several versions of the single-CPU Particle Computer have been developed, giving priority to miniaturization over flexibility. Hence, the Particle Computer design concept differs from that of PAVENET. In addition, although the Particle Computer is designed for an optimized original MAC protocol and a routing protocol, it is not designed for easy development of other communication protocols.

MICA Mote [4] and TinyOS [11], which originated as part of the Smart Dust project [9] at the University of California, Berkley, are widely used by researchers as a wireless sensor network testbeds [2), 27)~29)]. MICA Mote consists of off-the-shelf commercial parts and was developed as a demonstrative experiment of wireless sensor networks using several sensor nodes. Therefore, MICA Mote attaches importance to the lowest cost development and has single-CPU architecture. TinyOS is tiny operating system by which to realize functions for wireless sensor networks in very limited resources [11]. TinyOS is developed using nesC [6]. nesC has the ability to realize a minimum of wireless sensor network technologies in limited computing resources, but sacrifices some functions, such as hard real-time transaction support, that are necessary in order to develop various sensor network technologies. In addition, although nesC is an extension of the C language, it has a peculiar syntax and so users must learn a new language.

MANTIS, which is a multi-threaded operating system for wireless sensor networking devices, is being developed at the University of Colorado [7]. The operating system has various functions: a simple cross-platform API, a remote shell for debugging and logging, an RF-based fine grain dynamic reprogramming system, and an original file system for wireless sensor networks [30]. However, MANTIS does not provide a network layering architecture. Functions to support the development of new wireless communication schemes are very important for immature sensor network technologies.

PAVENET and related systems, such as Particle Computer, MICA Mote, TinyOS, and MANTIS, differ with respect to dual-CPU architecture or single- CPU architecture. The single-CPU architecture is unsuitable for use in situations in which various elemental functions must be verified.

BTnode has also been developed as a part of the Smart-Its project in ETH Zurich [31]. BTnode is an autonomous wireless communication and computing platform that is based on Bluetooth radio and a microcontroller. BTnode serves as a demonstration platform for research in mobile and adhoc connected networks (MANETs) and distributed sensor networks.

WinsNG sensor nodes were developed by Sensoria Corporation, under the DARPA SenseIT program [32]. WinsNG uses an SH4 CPU, Linux (as an operating system), GPS, seismic and acoustic sensors, and 802.11 radio.

BTnode and WinsNG use commercial radio such as Bluetooth and 802.11. This usage of existing radio approaches releases developers from considering wireless communication details, but the applications that can be developed using the testbeds are limited by the characteristics of the wireless radio. For example, Bluetooth requires a long time to establish connections between nodes. These restrictions are problematic because nobody can predict the assumptions used in future applications of wireless sensor networks.

6. Conclusion

In the present paper, we described the design and implementation of a wireless sensor network testbed called PAVENET. PAVENET has U^3, U^3 SDK, and basenode software. In order to support the development of wireless sensor network technologies, PAVENET has four characteristic features: hardware level modularization, dual-CPU architecture, hard real-time transaction support, and network layering APIs. We have verified the applicability of PAVENET by constructing a number of applications, including ANtennary THings, a traffic line detection system, and a MAC protocol performance measurement system.

References

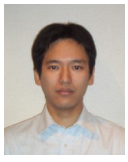
1) W. Heinzelman, A. Chandrakasan and H. Balakrishnan: "Energy-Efficient Communication Protocol for Wireless Microsensor Networks", Proceedings of the 33rd Annual Hawaii International Conference on System Sciences (HICSS'00), Maui, Hawaii, USA (2000).
2) W. Ye, J. Heidemann and D. Estrin: "An energy-efficient MAC protocol for wireless sensor networks", Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'02), New York, New York, pp. 1567–1576 (2002).
3) T. van Dam and K. Langendoen: "An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks", Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems (SenSys'03), Los Angeles, California (2003).
4) J. Hill and D. Culler: "MICA: A Wireless Platform For Deeply Embedded Networks", IEEE Micro, Vol. 22, pp. 12–24 (2002).
5) L. E. Homquist, H.-W. Gellersen, G. Kortuem, S. Antifakos, F. Michahelles, B. Schiele, M. Beigl and R. Maze: "Building Intelligent Environments with Smart-Its", IEEE Computer Graphics and Applications, Vol. 24, pp. 56–64 (2004).
6) D. Gay, P. Levis and R. von Behren: "The nesC Language: A Holistic Approach to Networked Embedded Systems", Proceedings of Conference on Programming Language Design and Implementation (PLDI'03), San Diego, California, ACM, pp. 1–11 (2003).
7) H. A. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker, J. Deng and R. Han: "MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms", Proceedings of the 2nd ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'03), San Diego, California, pp. 50–59 (2003).
8) T. Kashima, S. Saruwatari, H. Morikawa and T. Aoyama: "A Bind Control Model For Real-space Programming in Ubiquitous Computing Environment", Adjunct Proceedings of the 6th International Conference on Ubiquitous Computing (UbiComp'04, poster), Nottingham, England (2004).
9) J. M. Kahn, R. H. Katz and K. Pister: "Next Century Challenges: Mobile Networking for Smart Dust", Proceedings of the 5th Annual ACM/IEEE Internation Conference on Mobile Computing and Networking (MobiCom'99), Seattle, Washington, ACM, pp. 483–492 (1999).
10) D. Estrin, R. Govindan, J. Heidemann and S. Kumar: "Next Century Challenges: Scalable Coordination in Sensor Networks", Proceedings of the 5th Annual International Conference on Mobile Computing and Networks (MobiCom'99), Seattle, Washington, ACM, pp. 263–270 (1999).
11) J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler and K. Pister: "System Architecture Directions for Networked Sensors", Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'00), Boston, Massachusetts, ACM, pp. 93–104 (2000).
12) "TinyOS Tutorial". http://webs.cs.berkeley.edu/tos/tinyos-1.x/doc/tutorial/.
13) UPnP Forum: "UPnP Device Architecture 1.0" (2003).
14) Sun Microsystems, Inc.: "Jini Architecture Specification" (2001).
15) J. Waldo: "The Jini Architecture for Network-Centric Computing", Communications of the ACM, Vol. 42, pp. 76–82 (1999).
16) J. C. Haartsen: "The Bluetooth Radio System", Personal Communications, Vol. 7IEEE, pp. 28–36 (2000).
17) M. Beigl, A. Krohn, T. Zimmer, C. Decker and P. Robinson: "AwareCon: Situation Aware Context Communication", Proceedings of the 5th International Conference on Ubiquitous Computing (UbiComp'03), Vol. 2864, Seattle, Washington, pp. 132–139 (2003).
18) Particle Team: "PARTiCLE WEB SITE". http://particle.teco.edu/.
19) T. Zimmer: "Towards a Better Understanding of Context Attributes", Proceedings of the 2nd IEEE International Conference on Pervasive Computing and Commnications (PerCom'04), Orland, Florida, IEEE, pp. 23–28 (2004).
20) C. Decker, M. Beigl, A. Krohn, U. Kubach and P. Robinson: "eSeal: A System for Enhanced Electronic Assertion of Authenticity and Integrity of Sealed Items", Proceedings of the 2nd International Conference on Pervasive Computing, Linz, Vienna (2004).
21) C. Decker and M. Beigl: "DigiClip: Activating Physical Documents", Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04), Tokyo, Japan, pp. 388–393 (2004). IWSAWC'04.
22) M. Beigl, A. Krohn, C. Decker, P. R. T. Zimmer, H. Gellersen and A. Schmidt: "Context Nuggets: A Smart-Its Game", Adjunct Proceedings of the 5th International Conference on Ubiquitous Computing (UbiComp'03, demo), Seattle, Washington (2003).
23) C. Decker, U. Kubach and M. Beigl: "Revealing the Retail Black Box by Interaction Sensing", Proceedings of the 23th International Conference on Distributed Computing Systems (ICDCS'03), Providence, Rhode Island (2003). IWSAWC'03.
24) P. Robinson and M. Beigl: "Trust Context Spaces: An Infrastructure for Pervasive Security in Context-Aware Environments", Proceedings of the 1st International Conference on Security in Pervasive Computing (SPC'03), Boppard, Germany (2003).
25) M. Beigl, P. Robinson, T. Zimmer and C. Decker: "Teaching a practical Ubicomp Course with Smart-Its", Adjuct Proceedings of the 4th International Conference on Ubiquitous Computing (UbiComp'02), Goteborg, Sweden, pp. 43–44 (2002).
26) L. E. Holmquist, F. Mattern, B. Schiele, P. Alahuhta,

M. Beigl and H.-W. Gellersen: "Smart-Its Friends: A Technique for Users to Easily Establish Connections between Smart Artefacts", Proceedings of the 3rd International Conference on Ubiquitous Computing (UbiComp'01), Atlanta, Georgia, pp. 116–122 (2001).
27) Y. Kawahara, T. Hayashi, H. Tamura, H. Morikawa and T. Aoyama: "A Context-Aware Content Delivery Service Using Off-the-shelf Sensors", Proceedings of the 2nd International Confernce on Mobile Systems, Applications, and Services (Mobisys'04, Poster Presentation), Boston, Massachusetts (2004).
28) R. Suzuki, K. Makimura, H. Saito and Y. Tobe: "Prototype of a Sensor Network with Moving Nodes", Proceedings of the 1st International Workshop on Networked Sensing Systems (INSS'04), Tokyo, Japan (2004).
29) A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler and J. Anderson: "Wireless Sensor Networks for Habitat Monitoring", Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02), Atlanta, Georgia (2002).
30) H. Dai, M. Neufeld and R. Han: "ELF: An Efficient Log-Structured Flash File System for Wireless Micro Sensor Nodes", Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys'04), Baltimore, Maryland (2004).
31) J. Beutel, O. Kasten, F. Mattern, K. Römer, F. Siegemund and L. Thiele: "Prototyping Wireless Sensor Network Applications with BTnodes", Proceedings of the 1st European Workshop on Wireless Sensor Networks (EWSN'04), Berlin, Germany, pp. 323–338 (2004).
32) G. J. Pottie and W. J. Kaiser: "Wireless Integrated Network Sensors", Communications of the ACM, Vol. 43, pp. 51–58 (2000).

Shunsuke Saruwatari

He received the B.S. degrees in Computer Science from the University of Electro-Communications, and M.S. degree in Frontier Informatics from the University of Tokyo. He is currently a Ph.D. student of the Department of Frontier Informatics at the University of Tokyo. His research interests are in the area of computer networks, distributed computing, wireless networks, embedded computer, and wireless sensor networks. He is a member of IEICE, and IPSJ.

Takuya Kashima



He received the B.E. and M.E. degrees in Information and Communication Engineering from the University of Tokyo. He is currently working at KDDI Corporation.

Masateru Minami

He received B.E. and M.E. from Shibaura Institute of Technology, and Dr. Eng. from the University of Tokyo in 1996, 1998 and 2001 respectively. He is currently an assistant professor at Shibaura Institute of Technology. His research interests include location systems and sensor networks for ubiquitous computing.

Hiroyuki Morikawa

He received the B.E., M.E., and Dr. Eng. degrees in Electrical Engineering from the University of Tokyo, Tokyo, Japan, in 1987, 1989, and 1992, respectively. He is currently an Associate Professor of the Department of Frontier Informatics at the University of Tokyo. From 1997 to 1998, he stayed in Columbia University as a visiting research associate. His research interests are in the area of computer networks, distributed computing, mobile computing, wireless networks, and network services. He served as Editor of Transactions of Institute of Electronics, Information and Communication Engineers (IEICE) and on the technical program committees of IEEE/ACM conferences and workshops. He is a member of IEEE, ACM, ISOC, IPSJ, and ITE.

Tomonori Aoyama

He received the B.E., M.E. and Dr. Eng. from the Univ. of Tokyo in 1967, 1969, and 1991 respectively. Since he joined NTT in 1969, he was engaged in R & D on various communication networks and systems in the NTT Labs. He stayed in MIT as a visiting scientist in 1973-1974. In 1994 he was appointed to Director of NTT Opto-Electronics Laboratories, and then in 1995 he became Director of NTT Optical Network Systems Labs. In 1997 he left NTT, and joined the University of Tokyo, and he is now Professor in the Department of Information and Communication Engineering there. He is IEEE Fellow. He is past President of IEICE Communication Society. He is co-author or co-editor of several books for digital signal processing, ATM broadband networks and optical fiber transmission systems.

Reprinted from Trans. of the SICE
Vol. E-S-1 No. 1 74/84 2005