Application-Centric Networking Framework for Wireless Sensor Nodes

Sukwon Choi Hojung Cha
Dept. of Computer Science, Yonsei University, Seoul, Korea
{sukwon,hjcha}@cs.yonsei.ac.kr

Abstract

Wireless sensor network technology has found diverse applications in numerous fields. As the networking technology is refined in many ways, the need for system modulation with effective performance becomes essential. A multitude of architectures, which includes system abstraction and layering, has been proposed to solve the need at the operating system level. However, previous efforts do not qualify for networking architecture required by sensor networking, since they are aimed at hardware abstraction or protocol-based layering. In this paper, we classify developers into kernel, network and application developers and propose a network architecture that enables those developers to program independently. Network stack is separated into three different layers; MLL, NSL, DNL. This three-layered architecture provides an effective programming environment to sensor network developers by minimizing modification of other layers and maximizing reusability of the networking module. To validate the proposed mechanism, we implemented and assessed the performance with a few network algorithms and applications, based on the RETOS, which supports a dynamic loadable kernel module.

1. Introduction

The WSN (Wireless Sensor Network) technologies have made it possible to distribute sensor nodes in numbers and to gather information and to perceive specific events. The technology has been applied to various areas such as construction [1], science [2], oceanography [3], and now it finds more applications. Various networking protocols have been proposed and implemented because WSN requires optimized protocols depending on the applications. As the number of protocols and applications grow, effective management becomes more necessary from the viewpoint of the operating system. Accordingly, research becomes active in the areas of system abstraction, layering and modularizing, and results in various works. Among the research is optimized hardware abstraction architecture [4]. The research includes the development of operating systems, such as SOS [5], Contiki [6], and RETOS [7], which can be modified partially and dynamically. SOS supports system modularization allowing programmers to implement network protocols and applications on the statically programmed kernel with dynamic loadable library. Contiki supports a three-layered architecture that enables the development of kernel-independent applications. SOS and Contiki do not distinguish between kernel and applications since they adopt single stack architecture. RETOS overcomes the aforementioned architectural deficiency, which is unclear separation of kernel and applications. RETOS not only separates kernel and applications, but supports a dynamic loadable kernel library, providing an advanced implementation environment.

A clear definition of modules is required for modularized operating systems to be useful. This is particularly important considering the limited resource availability in WSN. In the current implementation environment, a developer takes the responsibility from developing a MAC protocol to programming applications. Thus, the module architecture relies heavily on the developer's programming style, and shows a lack of both system efficiency and module reusability. For instance, the primary usage of the modules of SOS and Contiki is debugging or modifying the implemented algorithms. This is caused by improper layering architecture, especially a lack of distinct layers for developing network protocols.

TCP-IP's wide usage originates from its clear layering and the layers' functionalities. This should apply to WSN as well. Existing networking protocols for WSN usually adopt a cross-layering architecture, which allows each layer to modify the functions of the layer below [8]. To provide an independent networking architecture B-MAC [9], for instance, adopts an independent MAC protocol architecture, and SP (Sensornet Protocol) [10] offers translucent link abstraction. In [11], data-centricity was defined as an important characteristic, and network layering was suggested for data fusion. These works partially succeeded in separating networking protocols from the rest of the kernel. However, a more concrete independent networking architecture is required. To give layers full independency, we should approach the architecture from the point of programming. WSN developers can be divided into kernel developers, network developers, and application developers. Kernel developers take responsibility for implementing core parts of operating systems so that the kernel fully utilizes the

hardware. Network developers implement various networking algorithms, and using those networking algorithms application developers program the required functions.

In this paper, we propose a layering architecture that provides a distinct and independent programming environment to those three classes of developers. The architecture includes both the advantages of traditional networking architecture and the resilience of a modularized operating system. In this architecture, application developers are able to build prototypes and to implement them easily. Network developers are able to program various networking algorithms in application- and kernel-independent way. For this purpose kernel consists of two parts; the static part and dynamic part. Appropriate protocol architectures will be given to those two parts of the kernel, allowing minimal modification to other layers and maximum reusability of network modules when implementing.

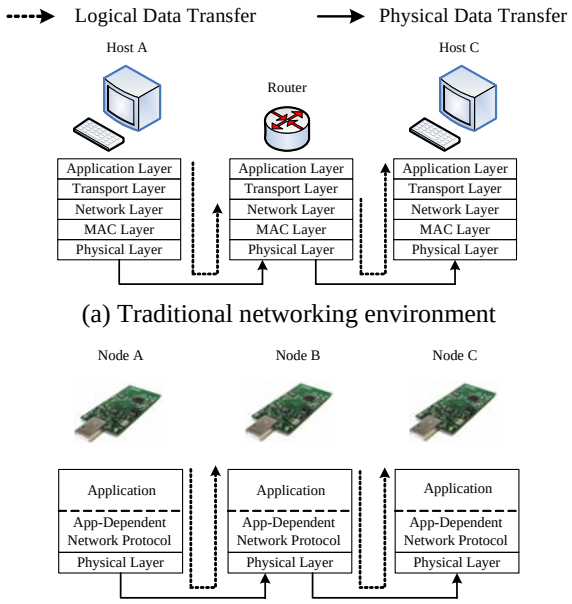
The rest of the paper is organized as follows. In Section 2, we review the RETOS operating system. In Section 3, we describe the proposed stack architecture. In Sections 4 and 5, various implementations of networking protocols and applications are given using the proposed network stack, and the performance is analyzed. Sections 6 and 7 show related works and conclude the paper.

2. Background

In this section we briefly discuss the elements that need to be considered when designing a networking architecture in WSN, and introduce RETOS which is the operating system used with our work.

2.1 WSN Network Architecture

WSN utilizes hardware with limited resources and adopts an ad-hoc multi-hop networking mechanism. These characteristics have made WSN use an application-dependent cross-layering architecture without separated layers. Figure 1 shows the difference in protocol architecture between a traditional network and a sensor network. Traditional networking architecture has hierarchical network layers, and each layer proceeds based on the function of the layer below. As shown in Figure 1 (A), only a network protocol and below are needed for the traditional routers to relay data packets, while, as shown in Figure 1 (B), each node of the sensor network functions as a router and the entire layers participate in data transmission. Multi-hop networking as well as in-network processing is required for sensor nodes to communicate with adjacent nodes. In-networking processing has different characteristics from multi-hop networking since the communication is targeted to the neighboring nodes. The majority of applications use both



(a) Traditional networking environment

(b) Sensor networking environment

Figure 1. Protocol stack in general network and sensor network

multi-hop and hop-to-hop networking for data transmission.

Considering these features, stack architecture for a sensor network should meet two requirements. First, independency of each layer should be guaranteed. Each layer should be able to choose a different networking protocol without affecting the other layers. Second, a higher layer should be able to receive the necessary information from the layer below. These are essential since different network protocols are adopted in various situations.

The layers of networking architecture should provide the necessary functions to relative developers, so that developers do not need to modify other layers. In this layering architecture, basic hop-to-hop communication is handled in lower layers, and end-to-end communication is handled in higher layers. Standardized API is defined to enable efficient information and data exchange between the layers, and appropriate API needs to be provided to the developers as well.

2.2 RETOS Overview

RETOS [7] is being developed to provide a reliable multi-threading environment and a dynamic Loadable Kernel Module (LKM) to users. RETOS supports POSIX 1003.1b real-time scheduler interface, and allows users to implement additional LKM using Standard C. RETOS adopts the dual mode and protection mechanism, and protects a system from the errors caused by application programs. RETOS has single kernel stack to minimize the

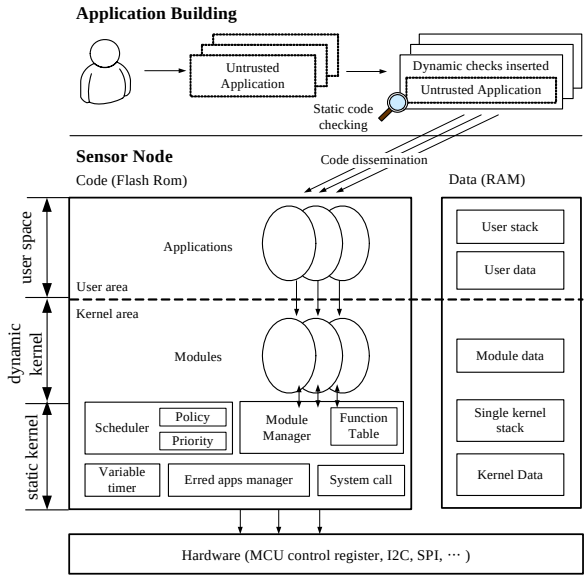


Figure 2. RETOS overview

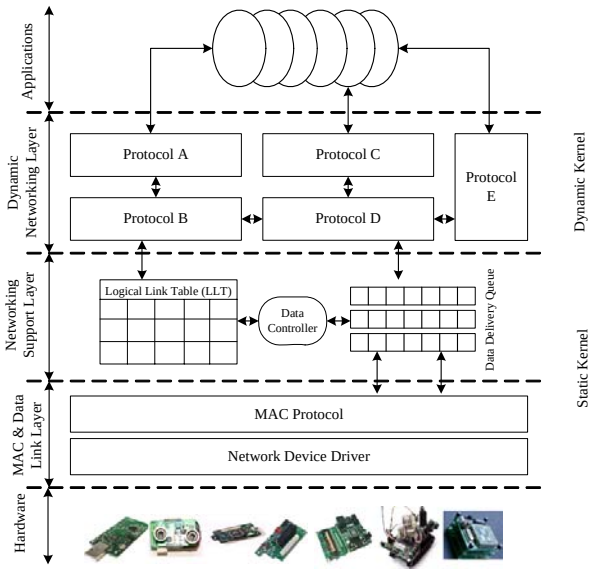


Figure 3. Proposed network stack

memory consumption imposed by adopted dual mode. RETOS is currently being implemented in MoteIV Telos. platform [12].

Sensor nodes are generally equipped with MPU, which does not have MMU. This lack of MMU prevents the memory addresses of the modularized kernel to be allocated when compiled. RETOS supports LKM architecture that overcomes this problem. The system allows users to choose kernel functionalities required by application programs, and the resulting modularized kernel saves resources and takes on a optimized form. Figure 2 describes RETOS with LKM. Application programs are based on the operating system after their feasibility is checked. Loaded application programs are given resources and managed by a scheduler. The operating system consists of static codes and dynamic kernel modules. Static codes have OS core functionalities and hardware-dependent codes. Dynamic codes contain shared libraries, which are available for application programs.

3. Application-Centric Networking Framework

3.1 Architecture Overview

Network layering architecture makes an easy development environment, but imposes overheads on developers since available resources are limited. In our work, we distinguish between performance-critical part and usability-critical part in terms of development. The performance-critical part is closely related to hardware and should be optimized at the device driver level by the kernel developer. The usability-critical part is classified into the network protocol part and application part. An optimized networking algorithm should be able to be implemented in the network protocol part, and data-driven programming, which enables sensor nodes to gather information, should be available in the application part. For this, this paper differentiates the static part of the kernel from the dynamic one. Network functionality in the static kernel is to maintain the connection to neighboring nodes and transmit the data packets to them. Implementation of networking algorithms and supporting application programs takes place in the dynamic part of the kernel.

Figure 3 shows three-layered architecture; MLL (MAC and Data Link Layer), NSL (Networking Support Layer), and DNL (Dynamic Network Layer). The static kernel contains MLL and NSL. The bottom layer MLL, which controls the network devices, manages the physical connection and transmits data packets. NSL supports logical connections, managing neighboring nodes and data transmission. DNL, which belongs to the dynamic part of the kernel, enables implementing network protocol algorithms and provides user API for easy development. In the proposed network architecture, three classes of developers can implement layer-specific programs, without interfering in others, and the usage of modularization of operating system becomes maximized. In particular, the RETOS loadable kernel module makes efficient network architecture possible. The following sections describe the detailed design of NSL and DNL.

3.2 Networking Support Layer

NSL, interposed between MLL and DNL, maintains the information of neighboring nodes and passes it to DNL.

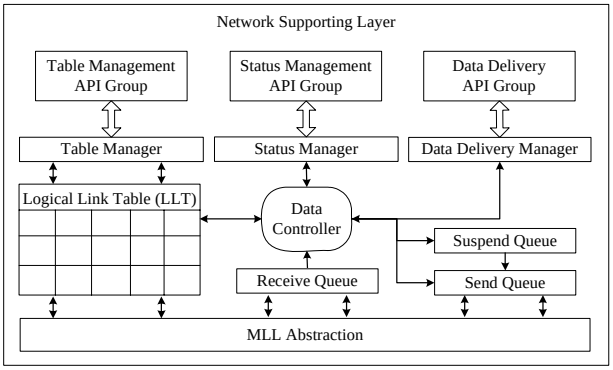


Figure 4. Networking Support Layer

Table 1. Elements of Logical Link Table (LLT)

Item	Description	Item	Description
Node ID	Node sequence number	Delivery Rate	Delivery success ratio
Position	Logical position	ACK/NACK	Use ack/nack or not
LQI	Link quality	Delivery Time	Packet delivery time
RSSI	Signal strength	Battery Level	Battery residual
CCA	Channel loading status	Valid	Node validate selection

The network algorithm used in DNL determines the destination node, and transmits the data to the target node. As shown in Figure 4, NSL consists of the Table Manager, Status Manager, Data Delivery Manager, and NSL API. The Table Manager manages Logical Link Table (LLT) which contains the information of neighboring nodes. LLT keeps the data between the nodes that can guarantee real-time, energy efficiency, and reliability (Table 1). To preserve LTT, the 2-way ADV-REQ handshake protocol [13], is used. The querying period is set to be periodic or non-periodic by network protocol developers by using API.

The Data Delivery Manager manages Send/Suspend/ Receive Queue for data transmission. Suspend Queue is added to Send/Receive Queue, allowing an individual router to function as a router in the proposed architecture. Suspend Queue is where received packets are suspended until the packets' next routing node is determined. The data transmission rate is changed using Suspend Queue, and this allows avoiding bottlenecks to be avoided and the traffic to be monitored. The Status Manager monitors the general status of the incoming data, the information of the specified neighboring nodes and the network queue. Then the Status Manger sends the information to DNL and sends the decision of DNL to NSL. The Status Manager also hands over the received packets to Suspend Queue, or transmits them directly to the neighboring nodes. NSL API, the interface of DNL and NSL, consists of the Table Management API Group, Status Monitoring API Group, and Data Delivery API Group so that NSL API controls three managers. Each API in the dynamic

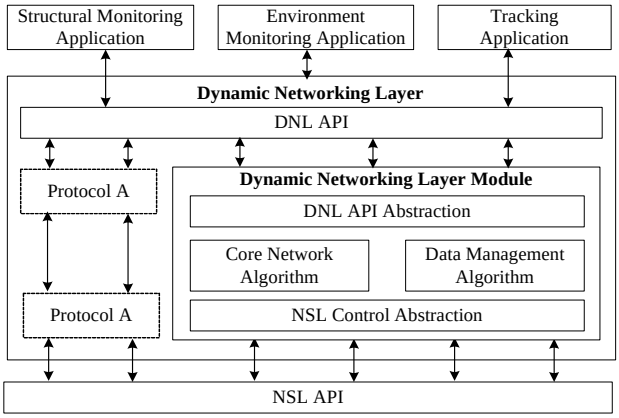


Figure 5. Dynamic Networking Layer

loadable kernel module is invoked as a kernel function.

3.3 Dynamic Networking Layer

In DNL, networking algorithms are implemented and packaged in a library using reconfigurable RETOS dynamic kernel modules. Protocols in the layer decide the destination of data packets, and send them to the appropriate node. The protocols can be built reliable or real-time depending on the applications and data types. Existing protocols can be implemented in this layer. For this, standardized API is defined in DNL, and implemented network algorithms provide generic API to users via DNL API. Various network algorithms are managed as a set of the library in the dynamic module. Standardized DNL API is readily available to users without causing any confusion. DNL modules are assigned dynamically depending on applications. For example, a structural monitoring application requires reliable data transmission, an environment monitoring application needs periodic data burst, and a tracking application detects events and sends the data in a limited time. They all need different network algorithms, and DNL provides the required modules to the applications.

Figure 5 describes the DNL module architecture. The DNL module keeps DNL API Abstraction to provide standardized DNL API. NSL API at the bottom of DNL maintains NSL Control Abstraction. The DNL module controls the information of neighboring nodes, network status information, and data transmission provided by NSL via NSL control abstraction. The core network algorithm and data management algorithm are implemented between DNL API abstraction and NSL control abstraction, making network protocols available to the DNL module. DNL API is defined in different ways since applications take various forms. In this paper, the concept of existing TCP/IP has been applied to the sensor network. To make applications independent from network protocols, users are allowed to choose a network

protocol; it should provide the required functionalities. This socket-like networking mechanism does not make usage of port numbers, but depends on the chosen network modules. Network protocols are implemented and encapsulated, and, by using them application developers are able to make independent applications regardless of the actual implementation of the network protocols.

4. Implementation

The proposed networking architecture has been implemented in the RETOS kernel. We have built MLL and NSL in the kernel and implemented DNL using various network algorithms with LKM, and evaluated the validity of the proposed network architecture.

Tmote Sky [12] uses the CC2420 [14] transceiver which offers a data rate of 250kbps in 2.4GHz. The hardware is compatible with IEEE 802.15.4 MAC. The data frames used in IEEE 802.15.4 of CC2420 are handled with packets, leaving little change to be controlled by software. That makes it difficult to a implement variety of MAC, but CC2420 provides several advantages. It handles the majority of the tasks at the hardware level, and offers a well-defined interface for software development leading to minimizing MLL in size and improving it in performance. CC2420 is equiped with the basic elements such as CCA and LQI, which are necessary to implement NSL. Their presence makes more efficient MLL than the software-based MAC such as S-MAC [9], B-MAC [19], or Z-MAC [15]. Assured that fundamental functionality such as MAC protocol will be supported by hardware in the future, we have implemented MLL out of a radio transceiver. MLL is implemented using RETOS Device Driver Architecture (R-DDA) for smooth connection to NSL.

MLL and NSL are both implemented in the static part of the kernel, but should be separated from each other to guarantee independency. MLL is involved with the hardware details, and has to be device-dependent. NSL should be device-independent since it manages neighboring nodes and transmitted data. We developed NSL and implemented the Neighbor Table, Network Queue, and API using a device-independent data structure. DNL uses a dynamic architecture and this causes more problems than MLL or NSL does. To implement DNL API Abstraction and NSL Control Abstraction, memory waste and dynamic linking should be handled. RETOS LKM solved these problems [7]. In the implementation, we used a single shared kernel stack to run the modules. In this case, however, module data, which are stored in the kernel stack can be corrupted during the context switching. To solve this problem, we blocked context switching while the module function runs.

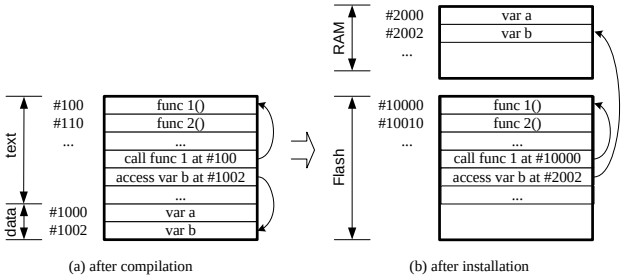


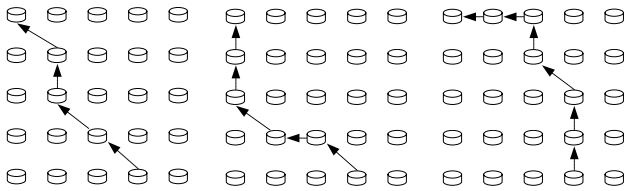
Figure 6. Relocation of DNL module

The dynamic module linking method is the next to be considered. Generally, MPU in a sensor node is a lack of MMU which translates a physical memory address into a virtual memory address and vice versa. In addition to the translation, MMU authenticates the access to the kernel mode. For these reasons operating systems, which use a loadable library, do not function properly without MMU. Relocation, as shown in Figure 6, was used to solve the problem. Absolute memory addresses are pre-defined when the source code is compiled, stored in the Relocation Table, and modified when the respective codes are loaded. In this case, the entire code image should be scanned and modified, but performance can be improved since memory access is available using absolute memory addresses.

The actual network protocol was developed on DNL. Reliability, real-time, and energy efficiency are the important characteristics for data transmission. They affect each others' performance, so maximizing all three features at the same time is difficult to achieve. The suggested network architecture should consider these characteristics of WSN. To show this we implemented respective network protocols for each feature. A reliable network protocol uses LQI measurement based routing algorithm [16]. Real-time network protocol adopts SPEED [17]. An energy-aware network protocol employs [18], which monitors the battery leftover and finds the optimized route, and was developed on NSL. These three network protocols were implemented using the RETOS loadable module. In the next section, we show that, while running the same application, a network protocol can be alternated with others without any modification of the application codes.

5. Evaluation

In general, sensor network applications can be classified into two fields. One includes periodic monitoring or event-responded monitoring, and various tracking programs that belong to the other. We implemented two different application programs to evaluate the performance of the suggested architecture; a simple environment monitoring application and an event-



(a) Reliable Network Protocol (b)Real-time Network Protocol (c) Energy-aware Protocol

Figure 7. Routing paths of RETOS-Surge according to three DNL protocols

responded tracking application. We chose the Surge, one of the representative applications built on TinyOS, for the environment monitoring application. Only the application part of the program has been ported, while the multi-hop routing algorithm has been excluded. We used RETOS-MPT [18] for the tracking application.

The performance of the applications has been evaluated using three DNL protocols. The architecture proposed in this paper provides an efficient programming environment, allowing minimum changes on the rest of the parts other than the application layer, and increased reusability. Thus, performance has been measured according to these independence and reusability.

5.1 Modularity and Adaptability

The reusability of modules has been tested by running two applications with different DNL modules. Since the applications were implemented using standardized DNL API, they should run by simply changing modules. Figure 7 shows the respective results of the three different DNL modules used, and illustrates the packets' routing paths. As shown in the figure, each routing path is dependent on the used DNL modules. This clarifies that DNL modules and applications can be developed and used separately. Hence various DNL modules and applications can be assembled with network protocols by needs.

To validate the independence of the layers, the structures of the network packets have been analyzed and are shown in Figure 8. Our architecture is distinguished from SP [10] in the fact that each layer is independent from the layer above. This is shown by analyzing the packets' independency between each layer and the one above. The structure of a packet is changed only in the DNL when a used module is alternated; meaning that a change of the packet does not affect packets in different layers. This shows that each layer functions independently and separately.

5.2 Overhead Analysis

The proposed network architecture minimizes the

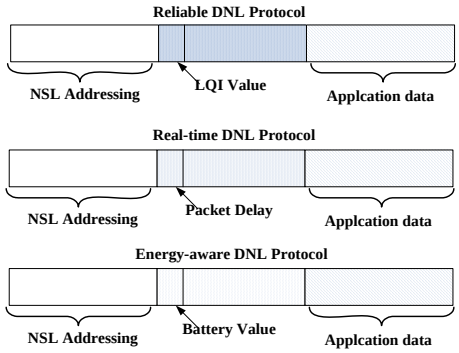


Figure 8. Send data packet composition

dynamic part to reduce performance overhead, and it should show the equivalent performance to those that do not use layering architecture. We compared RETOS-Surge with NSL and DNL to the counterpart without the layers, and observed overheads by measuring data transfer time in one-hop and multi-hop topologies. Data transfer time in the one-hop topology has been observed to check if it is stable and does not cause unreasonable overheads. Figure 9 shows that applications that do not make use of NSL have noticeable variation in data transfer time. However, the proposed architecture shows a faster and more stable data transfer time than its counterpart. The architecture that does not adopt layering principle has no proper queue control mechanism, and is affected by the nodes' current status, causing irregular transmission. On the other hand, our architecture guarantees fast and stable transmission since the queue control mechanism and the data transmission between nodes are conducted only within the NSL of the kernel.

The proposed architecture must show higher performance at data transmission than the ones not utilizing network architecture, because the proposed architecture only relays multi-hop data regardless of the

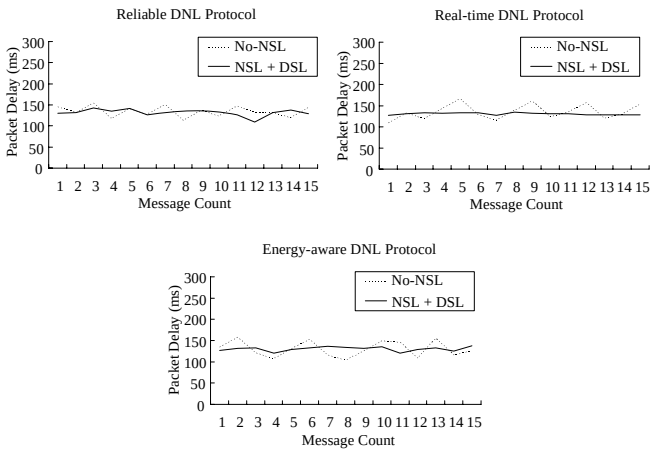


Figure 9. One-hop data transmission

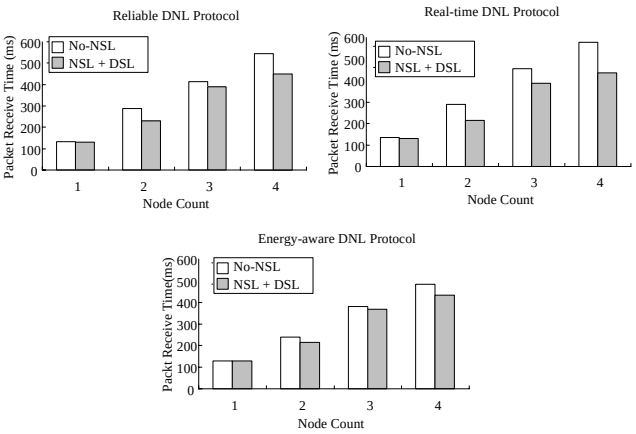


Figure 10. Multi-hop data transmission

Table 2. Code size for RETOS-Surge

	TinyOS (bytes)		RETOS Kernel (bytes)		RETOS Library + App (bytes)		RETOS Total (bytes)	
	ROM	RAM	ROM	RAM	ROM	RAM	ROM	RAM
Reliable DNL	17,036	668	20,464	1,052	1,003	243	21,467	1,295
Real-time DNL	24,046	1,522			2,731	317	23,195	1,369
Energy-aware DNL	18,201	1,800			2,473	223	22,937	1,275

Table 3. Code size for RETOS-MPT

	TinyOS (bytes)		RETOS Kernel (bytes)		RETOS Library + App (bytes)		RETOS Total (bytes)	
	ROM	RAM	ROM	RAM	ROM	ROM	RAM	ROM
Reliable DNL	20,944	597	20,464	1,052	1,942	262	22,406	1,314
Real-time DNL	26,944	1,743			3,123	351	23,587	1,403
Energy-aware DNL	20,253	1,943			2,987	278	23,451	1,330

packets' source or their destination. We observed multi-hop data transmission time, and the result is shown in Figure 10. As the number of hops increases, our architecture shows better performance than the one that does not as the number of hops increases. When network architecture is not used, the whole data packet is checked to determine the next hop. However in our architecture, fast data transmission is possible since switching overhead does not occur and data packets are sent directly in NSL.

In the sensor network environment, not only the performance of applications but also the size of codes or data is also important. The proposed dynamic architecture makes the size of codes larger than others do. We measured the size and it is shown in Table 2 and 3. As shown in the tables, the kernel, library, and application are run using 30k or less flash memory and 2k or less RAM. Although this size is bigger than TinyOS, it is acceptable considering the advantages of the dynamic architecture.

6. Related Work

The related work can be classified into three categories: network abstraction for WSN, kernel-independent application development, and application-independent implementation environment development. Network abstraction includes Ye [19], Dunkels [20], Kumar [11], and Polastre [10]. Ye makes an effort to implement WSN protocols in the traditional layered model, but succeeds only in the MAC layer. Dunkels implemented TCP/IP stack in 8-bit AVR MCU, however the characteristics are more suitable to a home network than to WSN. Kumar suggested a data centric network stack, but it had many obstacles to be deployed in real situations. It is not suitable for some application fields such as event detection, and independency of the layers is not guaranteed. [10] divided WSN protocols into the link layer and the rest, and bundled them using sensornet protocol (SP). However, these works do not fully adopt a network stack, but use a data structure to manage various protocols. Moreover, they do not classify application and network protocols. These features make these works more appropriate to an independent-image operating system such as TinyOS.

There are several works that try to separate kernel and application, and these include SOS [5], Contiki [6], and RETOS [7]. SOS divides a kernel into static part and dynamic part, and the latter is used to implement application. The message mechanism is used to enable communication between the static part and the dynamic part. For this, it suggests the message mechanism to enable communication between the static part and the dynamic part, becoming a successful operating system for WSN. Contiki separates the process from the static kernel, and uses the process to implement application. Threads are enabled within a process, and threads communicate with the kernel via messages. This is applicable to modularization since the development of the application is possible independent of kernel, however, the separation of the memory use is still imperfect. RETOS not only separates the kernel from application, but also supports LKM. The detailed work on RETOS is found in [7]. An application-independent development environment finds various works. Some of the most representative research fields are script programming and query programming based on middleware [21]. These methods make programming easy and the length of codes short, but use of various network protocols difficult. Whitehouse [22] and Welsh [23] suggest an easy application programming method, providing abstract API to users. However, applications are implemented on a static network protocol, and they are suitable only to certain areas.

7. Conclusion

The progress of WSN technology sees no limits, and the advent of devices with more computing power leads to the need of better WSN development environments. This paper offers a new WSN development environment, proposing network architecture that guarantees the efficiency and performance of development from the developers' point of view. The architecture holds the advantages of a static network layer of the traditional general purpose OS, and meets the diversity that is needed in WSN. By guaranteeing the independency of each layer, we increase the reusability of network protocols and provide agility and ease to the development environment. We first ported RETOS kernel to and implemented MLL and NSL. On them we developed several DNL protocols and examined the validity of the proposed architecture. The results show that efficient network programming is possible without any degradation of performance. With this result, we are assured that the proposed network protocol architecture is suitable for the use in a WSN, and suggest a guideline for modularized OS such as RETOS.

Future work will be extended to the validation of the architectures with SOS and Contiki. Additionally, we will build a library with various existing network protocols, and improve the performance of our architecture by developing a neighbor nodes managing method and a queue managing method.

Acknowledgements

The authors would like to thank Mr. Hee Tae Jung, of Yonsei University, for the help provided in the preparation of this paper. This work was supported by the National Research Laboratory (NRL) program of the Korea Science and Engineering Foundation (2005-01352), and the ITRC programs (MMRC) of IITA, Korea.

References

[1] N. Xu, S. Rangwala, K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan and D. Estrin., "A Wireless Sensor Network for Structural Monitoring," *In Proceedings of the ACM Conference on Embedded Networked Sensor Systems*, Baltimore, USA, November 2004.

[2] G. Werner-Allen, K. Lorincz, M.C. Ruiz, O. Marcillo, J.B. Johnson, J.M. Lees, M. Welsh, "Deploying a wireless sensor network on an active volcano," *IEEE Internet Computing*, accepted, in press.

[3] I, Vasilescu, K, Kotay, D, Rus, P. Corke, M. Dunbabin, "Data Collection, Storage and Retrieval with an Underwater Optical and Acoustical Sensor Network," *In Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys 2005)*, San Diego, USA November 2005.

[4] V. Handziski, J.Polastre, J.H.Hauer, C.Sharp, A.Wolisz and D.Culler, "Flexible Hardware Abstraction for Wireless Sensor Networks," *in Proceedings of the 2nd European Workshop on Wireless Sensor Networks (EWSN 2005)*, Istanbul, Turkey, January 2005.

[5] C. Han, R. K. Rengaswamy, R. Shea, E. Kohler and M. Srivastava, "SOS: A dynamic operating system for sensor networks," *Proceedings of the Third International Conference on Mobile Systems, Applications, And Services*, Seattle, USA, June 2005.

[6] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors," *In Proceedings of the First IEEE Workshop on Embedded Networked Sensors 2004 (IEEE EmNetS-I)*, Florida, USA, November 2004.

[7] H. Kim and H. Cha, "Towards a Reliable Operating System for Wireless Sensor Networks," *In Proceedings of the 2006 USENIX Annual Technical Conference*, Massachusetts, USA, May 2006.

[8] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient mac protocol for wireless sensor networks," *In Proceedings of the 21st International Annual Joint Conference of the IEEE Computer and Communications Societies*, New York, June 2002.

[9] J. Polastre, J. Hill, D. Culler, " Versatile Low Power Media Access for Wireless Sensor Networks," *In Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Maryland, USA, November 2004.

[10] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, I. Stoica, "A Unifying Link Abstraction for Wireless Sensor Networks," *In Proceedings of the Third ACM Conference on Embedded Networked Sensor Systems (SenSys 2005)*, San Diego, USA November 2005.

[11] R. Kumar, S. PalChaudhuri , D. Johnson , *Umakishore Ramachandran' Network Stack Architecture for Future Sensors*, Rice University, Computer Science, Technical Report, TR04-447

[12] MoteIV Co. Ltd., http://www.moteiv.com/.

[13] W. Heinzelman, J. Kulik, and H. Balakrishnan, "Adaptive Protocols for Information Dissemination in Wireless Sensor Networks," *In* Proceedings of the 5th ACM/IEEE Mobicom Conference, Seattle, USA, August 1999.

[14] Chipcon Inc., http://www.chipcon.com/.

[15] I. Rhee, A. Warrier, M. Aia, and J. Min, "Z-MAC: a Hybrid MAC for Wireless Sensor Networks," *In Proceedings of the Third ACM Conference on Embedded Networked Sensor Systems (SenSys 2005)*, San Diego, USA, Nov 2005.

[16] A. Woo, T. Tong, and D. Culler, "Taming the Underlying Challenges of Reliable Multhop Routing in Sensor Networks," *In Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, LA, USA, November 2003.

[17] T. He, J. A. Stankovic, C. Lu, and T. F. Abdelzaher, "SPEED: A Stateless Protocol for Real-Time Communication in Sensor Networks," *International Conference on Distributed Computing Systems (ICDCS 2003)*, Providence, RI, May 2003.

[18] W. Jung, S. Shin, S. Choi, H. Cha, "Reducing Congestion in Real-Time Multi-Party Tracking Sensor Network Application," *First International Workshop on RFID and Ubiquitous Sensor Networks*, Nagasaki, Japan, December 2005.

[19] W. Ye, J. Heidemann, D. Estrin, *A Flexible and Reliable Radio Communication Stack on Motes*, USC/ISI Technical Report ISI-TR-565.

[20] Adam Dunkels. Full TCP/IP for 8 Bit Architectures. *In Proceedings of the First ACM/Usenix International Conference on Mobile Systems, Applications and Services (MobiSys)*, San Francisco, USA, May 2003.

[21] Yong Yao and J. E. Gehrke, "Query Processing in Sensor Networks," *In Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR 2003)*, California, USA, January 2003.

[22] K. Whitehouse, C. Sharp, E. Brewer and D. Culler, "Hood: a Neighborhood Abstraction for Sensor Networks," *In Proceedings of ACM International Conference on Mobile Systems, Applications, and Services (MobiSys '04)*. Massachusetts, June, 2004.

[23] M. Welsh and G. Mainland, "Programming Sensor Networks Using Abstract Regions," In *Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04)*, California, USA, March 2004.