A Reconfiguration Method for Preserving Network Bandwidth and Nodes Energy of Wireless Sensor Networks

Hyunjun Jung[1], Dongwon Jeong[2], Byung-Won On[2] and Doo-Kwon Baik[1]
[1] Department of Computer and Radio Communications Engineering, Korea University
Anam-dong, Seongbuk-gu, Seoul 136-701, Korea
[e-mail: darkspen@korea.ac.kr, baikdk@korea.ac.kr]
[2] Department of Statistics and Computer Science, Kunsan National University
Miryong-dong, Gunsan-si, Jeollabuk-do 573-701, Korea
[e-mail: djeong@kunsan.ac.kr, bwon@kunsan.ac.kr]
*Corresponding author: Doo-Kwon Baik and Dongwon Jeong

Received August 20, 2015; revised November 18, 2015; accepted March 9, 2016; published May 31, 2016

Abstract

In Wireless Sensor Networks (WSNs) and even in the Internet of Things (IoT) ecosystem, the reconfiguration of sensor variables is an important problem when the role of a system (or application) program's sensor nodes needs to be adjusted in a particular situation. For example, the outdoor temperature in a volcanic zone, which is usually updated in a system every 10 s, should be updated every 1 s during an emergency situation. To solve this problem, this paper proposes a novel approach based on changing only a set of sensor variables in a part of a program, rather than modifying the entire program, in order to reduce both network congestion and the sensor nodes' battery consumption. To validate our approach, we demonstrate an implementation of a proof-of-concept prototype system and also present results of comparative studies showing the performance and effectiveness of our proposed method.

Keywords: sensor variable, wireless sensor network (WSN) application, reconfiguration

http://dx.doi.org/10.3837/tiis.2016.05.013 **ISSN : 1976-7277**

1. Introduction

The Internet of Things (IoT) constitutes the interconnection of various embedded computing devices, such as RFIDs, sensors, actuators, mobile phones, smart cars, and electronic devices, via the Internet. Tiny computers embedded in physical objects (or things) collaborate with other connected devices in the vicinity to provide intelligent services [1][2]. For example, a car driver would like to know the location of an available parking space in advance so that he or she can save his or her valuable time. In particular, Wireless Sensor Networks (WSNs) constitute a physical infrastructure that needs to embody the IoT ecosystem. In every WSN, there exist a sink node and a sensor network consisting of sensor nodes. The sink node is a type of sensor node that collects various sensing data, e.g., humidity, temperature, sound, pressure, motion, and vibration, from the sensor nodes in the sensor network. Then, based on these sensing data, a new IoT application can improve its quality of service (QoS) and mine more useful information by combining multiple heterogeneous sources, such as sensor networks, in addition to cellular networks, in an IoT environment.

First, to demonstrate the need for sensor network systems, we now show a real-world example. According to a news reports in Korea, 2 people were killed and at least 68 were injured in a pileup involving approximately 100 vehicles in foggy weather on the Incheon bridge, which connects the Incheon International Airport to the outskirts of Incheon. If the bridge had been equipped with an intelligent sensor network system, such as the Structural Health Monitoring system[1], which was designed, implemented, deployed, and tested on the 4200 ft. long main span and the south tower of the Golden Gate Bridge, such a serious traffic accident could have been prevented. For instance, many sensor nodes attached to the Incheon bridge frequently gather varied information concerning foggy, windy, or surveillance states, and the time interval (or named sensing period) for collecting sensing data can be reset. The collected data, e.g., those related to a foggy state, are stored in a sensor variable. Such a sensor variable has one of two binary values, 0 or 1: a value of 1 if there is fog around the bridge, and 0 otherwise. An example of a reconfigurable sensor variable is the sensing period. Let us suppose that the sensing period is initially set to 30 s. Since frequent updates of the sensor variable can deplete batteries, this is the update interval usually set for the surveillance state. However, in the case of an emergency, the sensing period needs to be shortened, e.g., from 30 s to 1 s, to reflect the conditions in the bridge's immediate vicinity. More importantly, when the values of reconfigurable sensor variables are changed in a sink node, the values corresponding to the sensor nodes should also be updated. This data synchronization problem is known as the reconfiguration problem in sensor networks. In this work, we first label each sensor variable either non-configurable or reconfigurable. For example, the sensor variable of the foggy state is labeled a non-configurable sensor variable. Meanwhile, if this sensor network system is connected to the cellular networks of the IoT ecosystem, additional information, such as that about sudden congestion in a particular area of the bridge, is combined with the values of all the sensor variables. Thus, a serious incident can be detected at its outset and an alarm can be given or car drivers who have just arrived at the bridge can be diverted.

However, if the reconfiguration problem is not handled in existing sensor networks, intelligent services cannot be expected in our daily life. Recall that the primary role of the sink node is to collect and store the values of the sensor variables arriving from the sensor nodes.

[1] Structural Health Monitoring of the Golden Gate Bridge (http://www.cs.berkeley.edu/~binetude/ggb/)

Users usually explore the values through monitoring the software of the sink node. If a user changes the value of a reconfigurable sensor variable, either the entire or part of the sensor nodes (called target sensor nodes) is not automatically updated according to this change. For instance, when the sensing period of a reconfigurable sensor variable for the surveillance state is changed from 30 s to 5 s, a user has to access the sensor node in person and manually change the value in the target sensor nodes. However, this is not a sensible idea if the sensor nodes are located within a danger zone. Alternatively, if the user wishes to update this value remotely, two methods are available. In the first, the entire source code is compiled, although the values of only a few reconfigurable sensor variables are altered in the sink node, and then, its binary codes are sent to the target sensor nodes. The disadvantages of this method are that the network congestion in the sensor network is increased and all the services have to be interrupted until all the execution files have been downloaded and then installed in the target sensor nodes. Hereafter, this method is called the full-image-based method. The second method is the component-based method. Typically, software usually consists of several components (or software modules), e.g., an operating system (OS) usually consists of a kernel module, networking module, etc. Thus, if the value of a reconfigurable sensor variable in a particular component is changed, e.g., the routing information is reset, the binary code corresponding to the component is transmitted only to target sensor nodes and is installed in them. In this case, the main shortcomings of the full-image-based method can be somewhat diminished. As would be expected, according to our experimental results, network congestion is mitigated in the component-based method much more effectively than in the full-image-based method. However, the component-based method still suffers from network congestion and battery requirement issues.

Therefore, to handle the limitations of these existing methods, in this paper, we propose a novel reconfiguration method in which the OS and application software contains a list of pointer variables, each of which contains an address designating its location in flash memory.[2] In the location, the value of each reconfigurable sensor variable is in fact stored. Therefore, although the value of a reconfigurable sensor variable is altered in the sink node, the proposed method simply updates the changed value in the pointer variable. Thus, it is no longer necessary to compile either the entire or part (i.e., components) of the source code in the sink node, and thus, the corresponding binary code does not need to be sent to target sensor nodes. In turn, this process does not give rise to serious network congestion. The main contributions of this study are as follows. (1) A prototype of the reconfiguration method in sensor networks was developed, and (2) the empirical study showed that our proposed method outperforms other methods in terms of the network congestion and battery issues that challenge sensor networks.

The rest of this paper is organized as follows: In Section 2, we summarize related work. In Section 3, our proposed reconfiguration method is presented in greater depth. Next, a prototype of the proposed method is described in Section 4 and the experimental results for three methods are discussed in detail in Section 5. Finally, in Section 6 the concluding remarks are presented, followed by our future research plan.

2. Related Work

A significant amount of research has been conducted on sensor network reconfiguration. Deluge [3] is the most popular application in the full-image-based method category. Deluge is

[2] Because flash memory is non-volatile memory, the data stored in flash memory are not lost when a sudden power outage occurs.

an application for replacing a monolithic binary image. The large size of a monolithic binary image (30–40 KB) incurs a high energy overhead for code transmission. Reinstallation of the full application also disrupts the running application, resulting in a loss of recent data and resources. Additionally, small frequent updates in the code, such as bug fixes, cause only minor differences between the new and old system images.

FlexCup [4] is a flexible code-updating mechanism that minimizes the energy consumed at each sensor node for installing arbitrary code changes. FlexCup requires a reboot after reconfiguration. OpenCOM [5] is a component model that can be explicitly tailored for diverse domains and environments. The component model is supported by a reflective runtime architecture that is itself built from components. OpenCOM is a generic model that was developed for resource-rich platforms. Gridkit [6] is sensor middleware that can be customized to suit different sensor application types, providing a reflective approach for a coordinated network-wide dynamic reconfiguration of sensor behavior. Figaro [7], a programming model supported by an efficient run-time system and distributed protocols, focuses on a code distribution algorithm for WSNs. The LooCI [8] component model imposes minimal additional overhead on developers, and uses an event-based binding model that enables them to model rich component interactions. Finally, RemoWare [9] mitigates the cost of post-deployment software updates on sensor nodes through the notion of in situ reconfigurability and provides a component-based programming abstraction to facilitate the development of dynamic WSN applications. Contiki [10] enables partial code updates through the linking and loading of native code in the ELF file format. The partial code updates require OS support.

Recently, the dynamic reconfiguration issue has attracted increasing research attention. Dynamic reconfiguration involves the automatic configuration of sensor nodes, and many dynamic reconfiguration methods have been proposed. In [11], a framework called Fennec Fox for dynamic reconfiguration to support heterogeneous applications was proposed. The framework comprises a runtime infrastructure and a programming language to specify the various WSN configurations. Leligou et al. [12] presented use cases that can exploit the reconfiguration feature. In this study, a sample application, an urban surveillance case based on a set of different sensors, was analyzed. The analysis identified communication modules and parameters that can be reconfigured. Hughes et al. [13] proposed an energy-aware software evolution model for WSNs. The model uses LooCI middleware and the authors proposed a generic model for calculating the energy cost of the reconfiguration in WSN. Yeh et al. [14] presented a framework for reconfigurable techniques on a WSN at the node and the network level. The framework develops a structure for optimization problems and classification of an energy optimization function. In [15], an approach called MoRE was proposed for developing autonomic systems in smart homes, reusing variability models at runtime to provide a semantic base for decision making. In this paper, the model-based reconfiguration engine to implement model-management operations was introdiced. Ortiz et al. [16] proposed a runtime variability mechanism that can modify the structural variability of a dynamic software product line. In this study, a super-type-enabled runtime variability mechanism was introduced and the previous feature models were extended. Mouronte et al. [17] proposed a high-level architecture that exploits runtime software variability techniques to manage the context-aware information dynamically. In [18], a family of configurable middleware with a flexible architecture, called FamiWare, was proposed. The middleware provides a single interface for accessing all those services that can operate in heterogeneous devices. The architecture uses a software product line. In [19], a family of middleware for a

WSN, called Family, that can be customized according to the imposed constraints was proposed. The middleware uses the software product line approach and feature model.

In addition, there are several issues related to WSNs, one of the most important of which is the collection, aggregation, lifetime enhancement and use of sensor data from various WSNs. Jung et al. [20][21], Zhang et al. [22] and Liu et al. [23] addressed this issue but it needs to be further investigated and improved..

There has been considerable research on the reconfiguration of WSNs, but several issues remain to be resolved. One of the main issues is that these approaches still suffer from network congestion and battery requirement issues. We propose a novel reconfiguration method based on sensor variables. The proposed method simply updates the changed value in the pointer variable.

3. Main Proposal

Fig. 1 shows an overview of the proposed method. For example, when a user changes the value of the reconfigurable sensor variable "sensing period" from 1 s to 5 s, the reconfiguration manager in the sink node creates a sensor variable packet (SVP), which includes the target sensor node identifier and the changed value of the reconfigurable sensor variable. Then, it sends this packet to the target sensor node. After the packet is received in the target sensor node, it is analyzed, and then, the value of the corresponding sensor variable in the non-volatile memory is changed from 1 s to 5 s. In our approach, not all the binary image data need to be sent to the target node. Instead, only a small packet is sent to update the corresponding sensor variable in the target sensor node. In addition, a changed valued is not lost when a sudden blackout occurs, because it is stored in non-volatile memory.

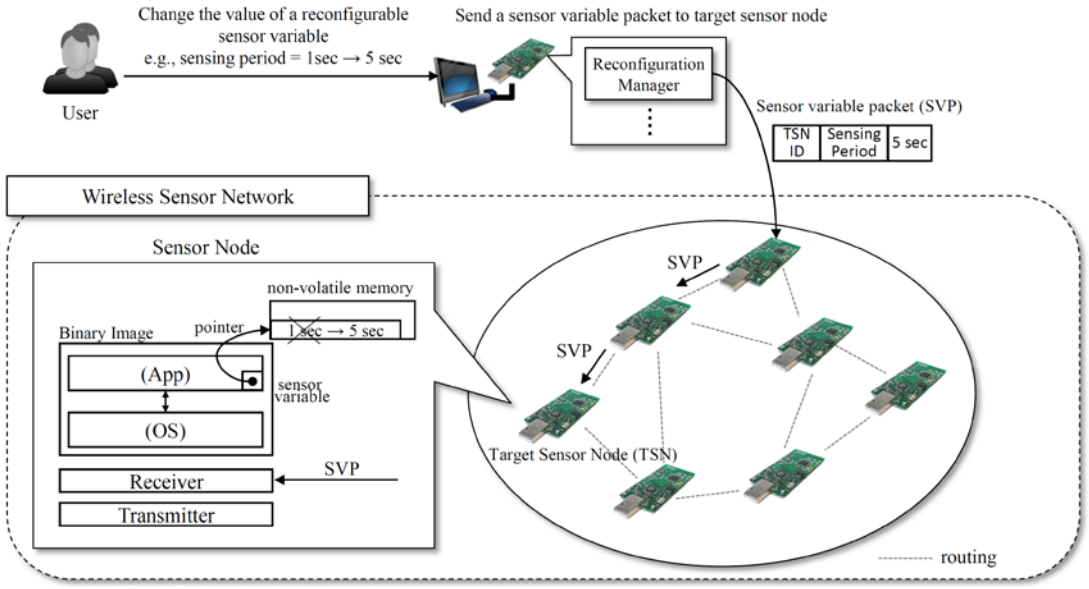

Fig. 1. Sensor variable-based reconfiguration method overview

3.1 Sensor Variable

As mentioned previously, the goal of this study was to devise a new reconfiguration method that supports a sensor variable-based reconfiguration. First, in our definition, a "sensor variable" contains the value that should be synchronized between the sink node and target sensor nodes. In other words, the value of the sink node should be always the same as that of the target sensor nodes. The sensor variables that are mentioned in ATaG [24][25][26] are similar to our sensor variable in that they represent the capability of a node and are used to determine the type of a node. Our approach is to modify the behavior by changing the sensor variable of the sensor node.

Sensor variables comprise a network, scheduler, device driver, sensor type, and other components. WSN application systems contain various sensor variables [27][28][29]. **Table 1** shows the reconfigurable sensor variable classification as defined in this paper and examples of sensor variables and descriptions according to category. The reconfigurable sensor variables are classified into four categories: network, scheduler, device driver, and sensor type. The network category includes sensor variables for network configuration, the scheduler category consists of sensor variables for schedule configuration, the device driver category consists of device sensor variables for sensor drivers, and the sensor type category includes sensor variables for sensors and actuators action.

Table 1. Sensor variable classification and description

Category	Sensor variable	Description (of sensor node)
Network	NodeID	Changes unique number for sensor node identification
	SensingPeriod	Changes sensing period value
	AdjacentActuatorNodeID, AssociationPermiStartNodeID, AssociationPermiEndNodeID, NextHopRoutingFirstNodeID, NextHopRoutingfirstNodeID, NextHopRoutingSeecondNodeID	Changes network identifiers relative to the sensor node
	RFChannel	Changes RF channel number
	Beacon_Enable	Changes Beacon uses
	ZigBeeRF	Changes ZigBee protocol type (e.g., simple, IEEE802.14.4, MAC, and StarMesh)
Scheduler	Scheduler_Enable	Changes Scheduler enable
	SchedulerType	Changes Scheduler type (e.g., None, FIFO, and PreemptionRR)
Device driver	EEPROM_Enable	Changes EEPROM module enable
	UartType	Change Uart type (e.g., NONE, printf-module, scanf-module, and scanf&printf)
	FlashMemory_Enable	Changes Flash memory module enable
	RSSI_Enable	Changes RSSI module enable
Sensor type	Sensor_Temperature_Enable	Changes Temperature module enable
	Sensor_LED_Enable	Changes LED sensor module enable
	Sensor_Gas_Enable	Changes Gas sensor module enable
	Sensor_Humidity_Enable	Changes Humidity module enable
	Sensor_Light_Enable	Changes Light module enable
	Sensor_Ultra_sonic_Enable	Changes Ultra sonic enable
	Sensor_Battery_Enable	Changes Battery enable

3.2 Code Description for Sensor Variable-based Reconfiguration

The code description of the sensor variable-based reconfiguration method uses sensor variable-based programming [30], which supports value updates and does not require a system reboot. A non-volatile memory of a sensor node enables a flash memory, e.g., EEPROM. Non-volatile memory has a storage area for non-volatile data, which are not lost even when the power is turned off suddenly. A reconfigurable sensor variable connects to the non-volatile memory. In the proposed method, we redesign OS and application software such that it includes a list of pointer variables, each of which contains an address designating its location in the memory. When the value of a reconfigurable sensor variable is changed in a sink, the value corresponding to a sensor node should also be updated. In the proposed method, the value of the reconfigurable sensor variable is in fact stored in the non-volatile memory of the sensor node.

Fig. 2 shows the logical structure connecting the sensor variable and the non-volatile memory. A sensor node consists of sensor variables, such as Node ID, Sensing_Period, and Humidity_Enable. The sensor variable values are stored in non-volatile memory by linking the corresponding code descriptions. When a sensor node changes its actions, the method proposed in this paper updates the sensor variable values by using the reconfiguration method. This feature supports a dynamic reconfiguration in WSNs.

Fig. 3 shows an example sensor variable-based code description in TinyOS [31]. The code description is connected to the reconfigurable sensor variable in a binary image and the non-volatile memory.

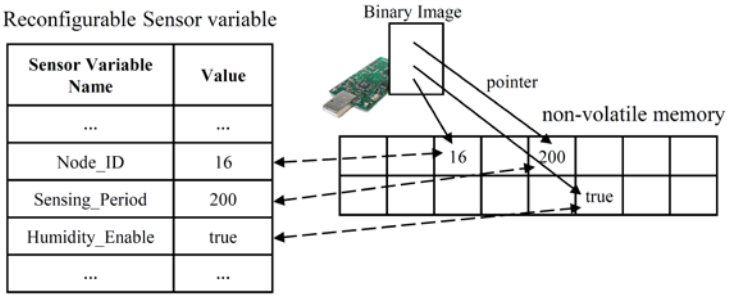

Fig. 2. Logical structure between sensor variable and non-volatile memory

```
Define a device set which has just set/get information as set_device.
Define a device information as device.
Define a device configuration as config_device.

begin
 if( set_device = "set" )
   switch(device)
     case "Node_ID"
                  setNodeID(config_device);
     case "Sensing_Period"
                  setSensingPeriod(config_device);
     case "Humidity_Enable"
                  setHumidityEnable(config_device);
   end switch;
 else if( set_device = "get" )
   switch(device)
     case "Node_ID"
                  config_device = getNodeID();
     case "Sensing_Period"
                  config_device = getSensingPeriod();
     case "Humidity_Enable"
                  config_device = getHumidityEnable();
   end switch;
 end if;
end;
```

Fig. 3. Sensor variable-based programming in TinyOS

3.3 Sensor Variable-based Reconfiguration Method

In this section, we describe the proposed method, which supports sensor variable-based reconfiguration. First, the sink node sends an initial communication packet to target sensor nodes to set the communication channel between the sink node and the target sensor nodes and then to hold the ongoing service of the target sensor nodes. Then, the sink node transmits an SVP to the sensor node for the reconfiguration. Each target sensor node updates the value of the reconfigurable sensor variable in the non-volatile memory.

Fig. 4 depicts an overview of the sensor variable-based reconfiguration method proposed in this paper. A user sends the reconfiguration information to the sink node. The reconfiguration information consists of the changing information of the target sensor node, i.e., the sensor node ID, sensor variable name, and value. The sink node checks the sensor variable list for validation of the reconfiguration information. The sensor variable list includes an available sensor variable list of the WSN and the unit of reconfiguration. The sink node creates SVPs using the reconfiguration information. The sensor node in general collects data and transmits the data to the corresponding sink node. If the sensor node executes the reconfiguration during sensor data collection, data collection errors may occur. Therefore, before executing the reconfiguration operation, the mode of the target sensor node changes to a waiting mode for error prevention. The sink node transmits the SVP after initial communication checking. The sensor node executes the reconfiguration operation using the SVP analysis.

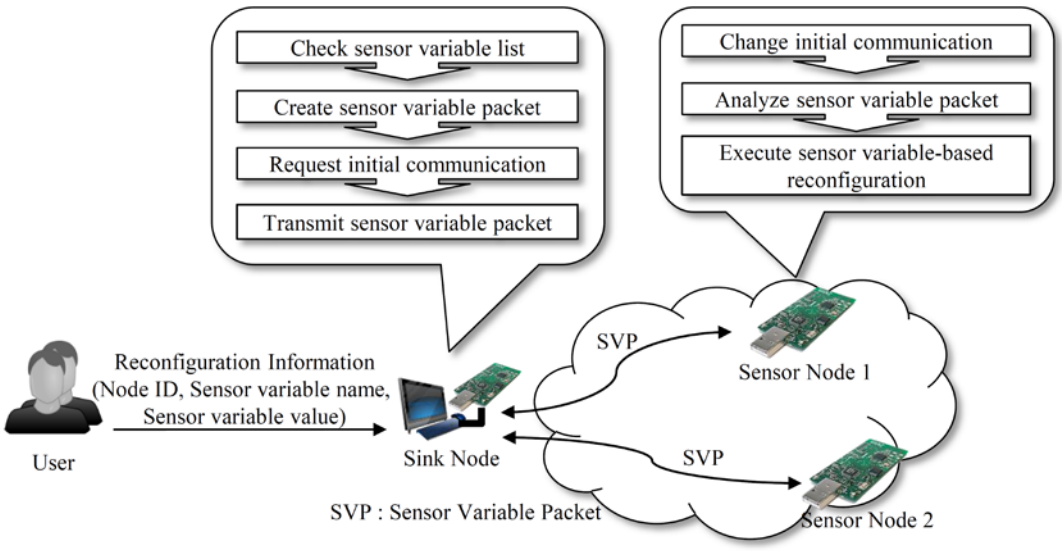

Fig. 4. Overall reconfiguration process in proposed method

Fig. 5 illustrates the structure of the sink node and sensor node. The following points summarize the module:

- **Selector:** The selector module selects the target node and the sensor variable.
- **Registry manager:** The registry manager module manages the sensor variable list of the WSN for validation of the reconfiguration information using the registry.
- **Sensor variable packet creator (SVPC):** The SVPC module generates the SVP. The packet includes the reconfiguration information.

- **Reconfiguration manager (RM):** The RM module manages the initial communication of the sensor node before a reconfiguration.
- **Sensor variable checker (SVC):** The SCV module manages the available sensor variable list of the WSN.
- **Packet analyzer (PA):** The PA module determines the packet types received from the nodes and analyzes the packet types. The packet types are related to the changing initial communication and executing reconfiguration.
- **Reconfiguration controller (RC):** The RC module executes the reconfiguration using non-volatile memory.
- **Sensor variable adaptor (SVA):** The SVA module changes the sensor variable values on the basis of the non-volatile memory.

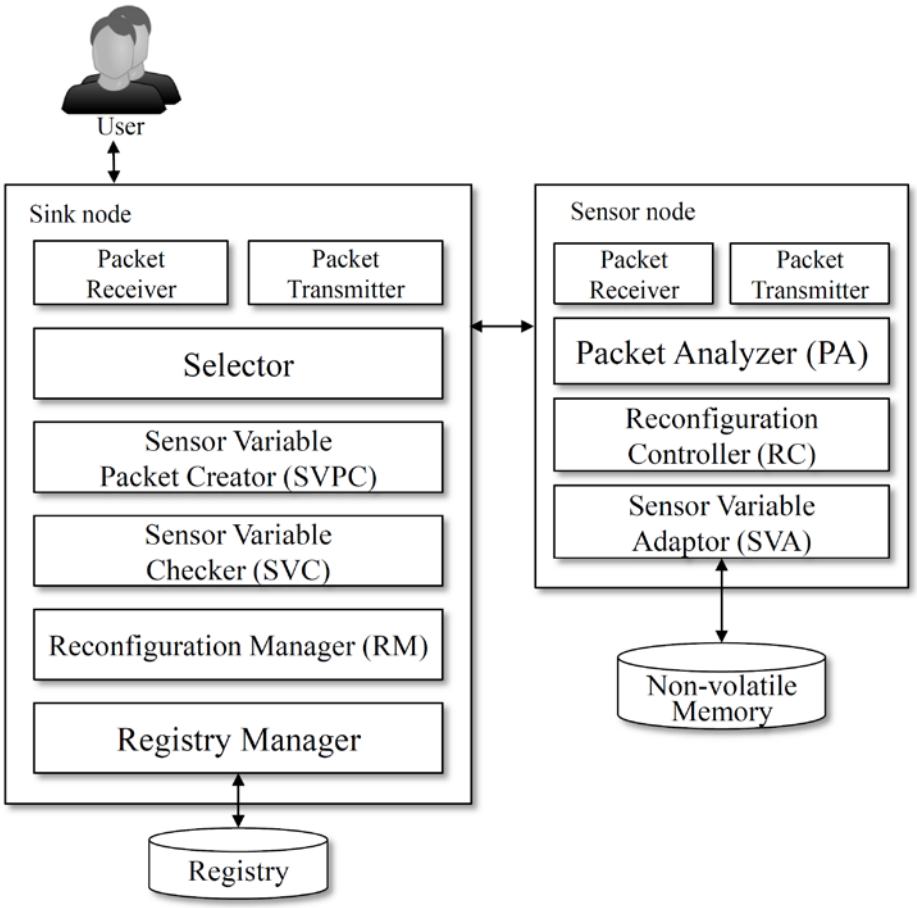

Fig. 5. Structure of the sink node and sensor node

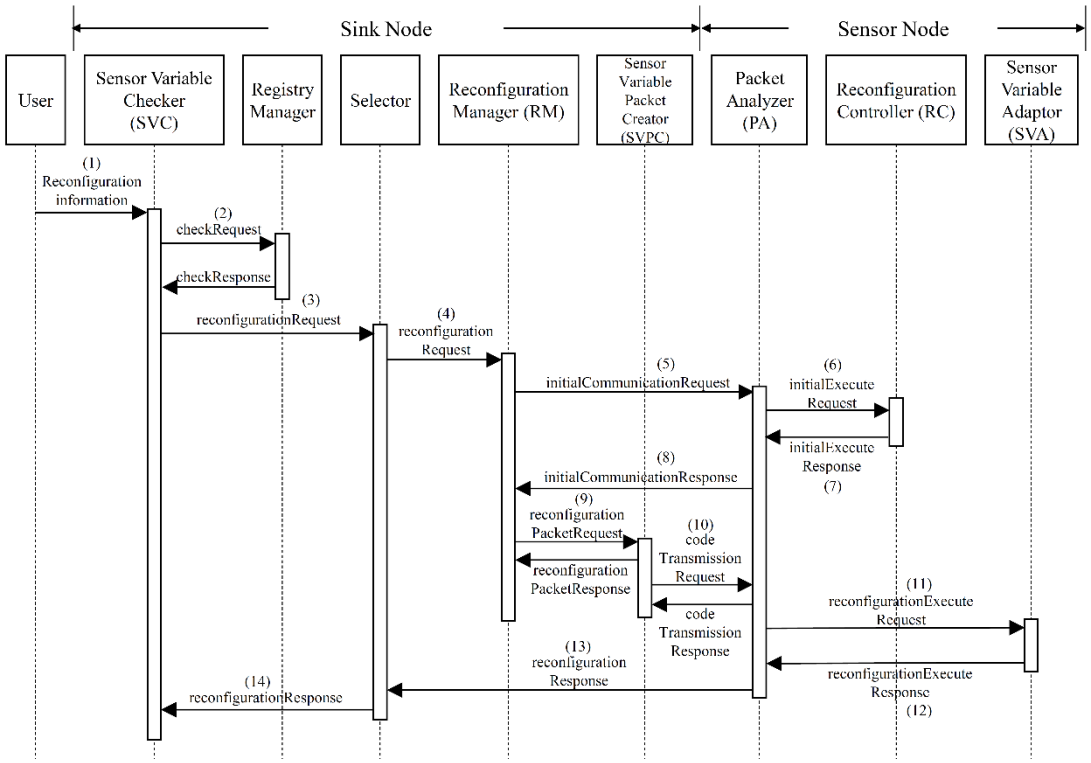

Fig. 6. Overall reconfiguration process: sequence diagram

Fig. 6 shows a sequence diagram of the overall proposed method process. The diagram shows the interaction between the modules. A user sends reconfiguration information for the sensor variable-based method (1). The reconfiguration information includes the sensor node ID, sensor variable name, and sensor variable value. The registry contains the sensor variable list of the WSN. The SVC checks the reconfiguration information using the registry manager (2). The selector selects the reconfiguration elements together with the SVC (3). The selector sends the result to the RM, which then executes a sensor variable-based reconfiguration (4). The RM requests a change in the initial communication at the target sensor node (5). The PA of the target sensor node analyzes the received packet and calls the RC, which changes the initial communication in the sensor node and sends the PA the result of the initial communication change (6) and (7). The PA transmits the result of the initial communication change to the RM (8). If the change result of the initial communication is true, the RM calls the APC (9). The SVPC creates a sensor variable-based reconfiguration packet, and transmits it to the target sensor node (10). The PA of the target sensor node analyzes the received the packet and calls the SVA (11). The SVA adapts the SVP for the reconfiguration and sends the PA the adaptation result (12). The PA transmits the result of the adaptation to the selector and the selector sends (13) the result to the SVC (14).

Fig. 7 shows the reconfiguration process for the sink node. The user sends any reconfiguration information for the target sensor node. The reconfiguration information includes the target sensor node ID, the sensor variable name, and the sensor variable value. The sink node receives the reconfiguration information from the user and checks the sensor variable list for validation of the reconfiguration information using the registry. The list includes a set of available sensor variable lists of all the sensor nodes and the unit of

reconfiguration. The sink node creates an SVP on the basis of the reconfiguration information. The SVP includes the target sensor node ID, the sensor variable name, and the sensor variable value. If there is no relevant reconfiguration information, the sink node requests new reconfiguration information from the user. The sink node requests the initial communication change before transmitting the SVP. The initial communication is ready for the reconfiguration execution for the target node. The sensor node in general collects sensor data and transmits the data to the sink node. If the sensor node executes the reconfiguration operation during sensor data collection, data collection errors may occur. The initial communication prevents sensor node errors during the reconfiguration. The sink node transmits the SVP after the initial communication checking is complete. Finally, the sink node transmits the SVP to the corresponding sensor node.

Fig. 8 shows the reconfiguration process for the sensor node. First, if the sensor node receives initial communication packets from the sink node, it changes its initial communication. The sink node waits to receive the SVP and execute sensor variable-based reconfiguration. Finally, the sensor node transmits the results of the reconfiguration to the sink node.

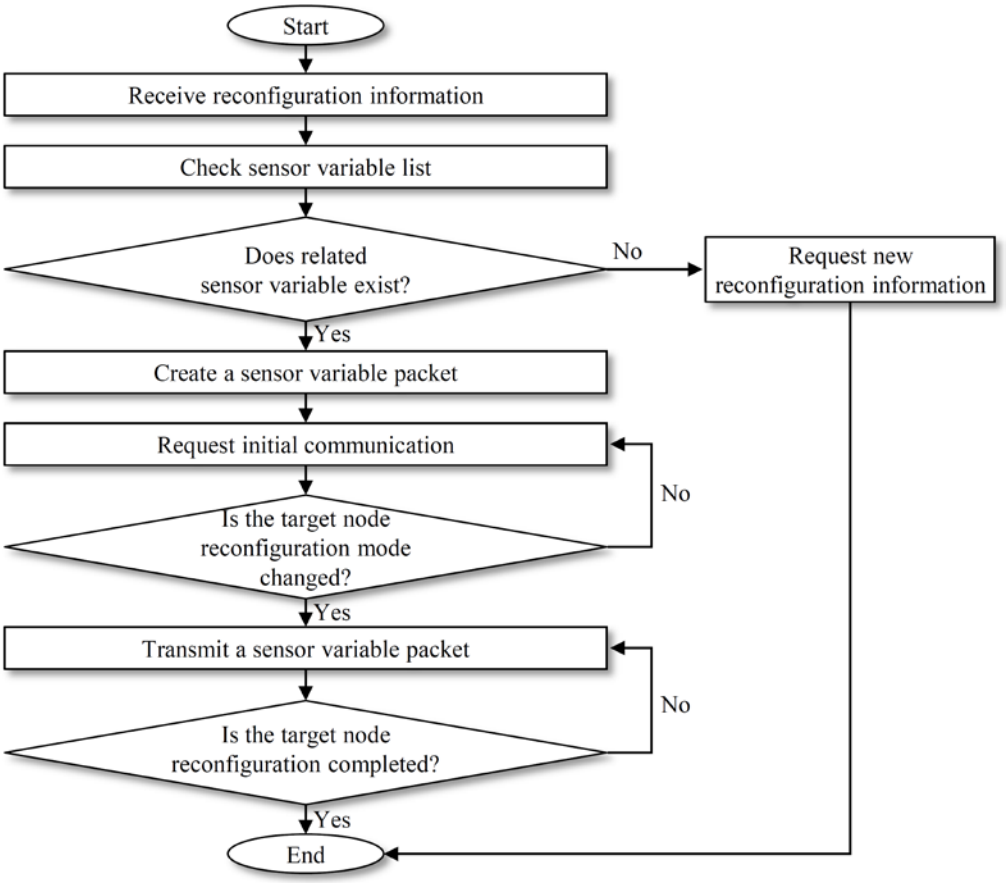

Fig. 7. Sink node process in the proposed method

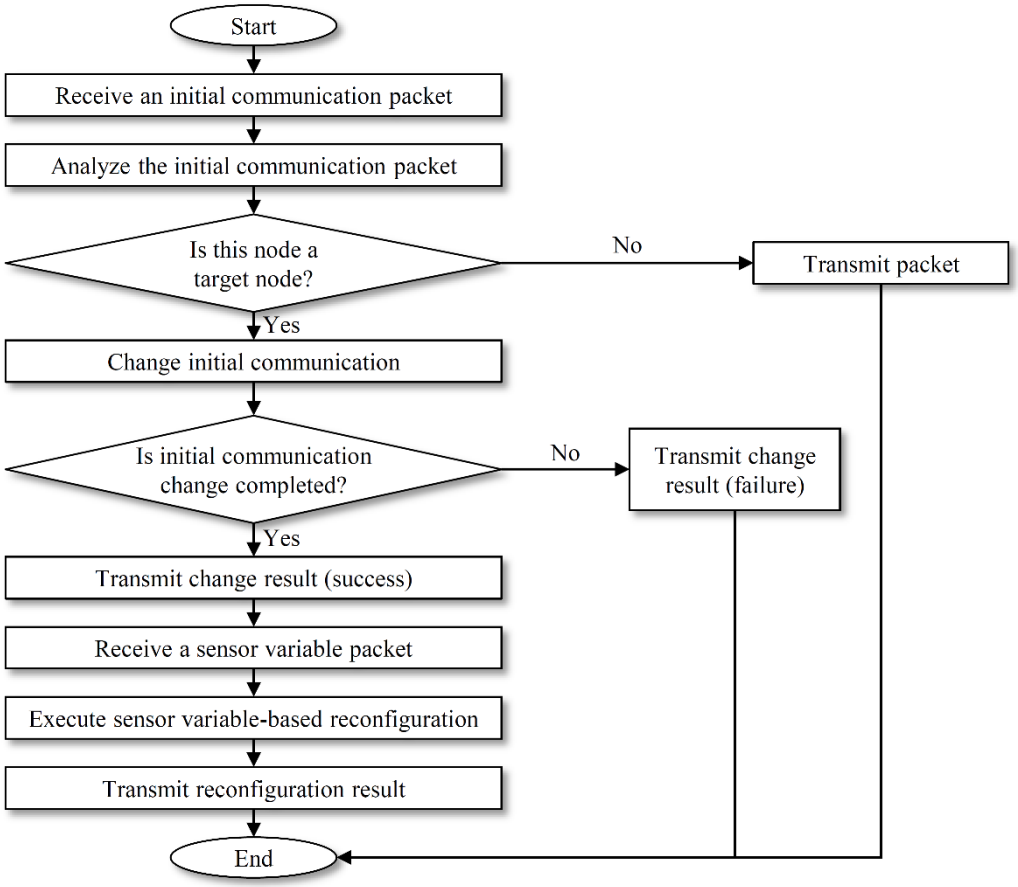

Fig. 8. Sensor node process in the proposed method

4. Implementation

This section describes a prototype of the sensor variable-based reconfiguration method proposed in this paper. The prototype was used to test the proposed method. We configured the nodes to use the 2.4 GHz band IEEE 802.15.4 physical layer and the beaconless mode action. The implementation used two TelosB nodes [32] running on TinyOS.

The prototype implementation used the flash memory of the non-volatile memory in the TelosB. The flash memory and EEPROM of the non-volatile memory enable the sensor node for sensor variable-based reconfiguration. Some parts of the flash memory and EEPROM can be read and written by the developer.

A sensor variable is connected to a source code with the flash memory. The proposed method changes the sensor variable values of the flash memory for changing the action. The reconfiguration information set, including the node ID, sensor variable name, and sensor variable value, is given to the prototype. The prototype implementation executes the changing operation for the sensor node.

Fig. 9 illustrates an implementation scenario to show the contributions of the proposed method. The sensor node supports the LED_Enable and Sensor_Temperature_Enable sensor variable. A user provides reconfiguration information containing the node ID, sensor variable name, and sensor variable value. The scenario comprises three steps for the action change in

the sensor node. In the first step, the user sends information about the LED sensor variable and Temperature sensor variable to the sink node. The sink node checks the sensor variable list and the sensor node creates an SVP. The packet includes the target sensor id, reconfigurable sensor variable name, and sensor variable value. The sink node requests an initial communication change from the target sensor node and transmits the SVP. The LED and Temperature elements of the target sensor node are deactivated. In other words, the LED light is turned off and the temperature sensor is not activated. In the second step, the LED action on the target sensor node is changed. The target sensor node is changed. After the change operation, the LED action is activated by using the LED_Enable sensor variable and the LED light is turned on. Finally, the temperature sensor action using the Sensor_Temperature_Enable sensor variable is changed as the activation status. In this scenario, in the proposed method only the SVP is requested for the reconfiguration.

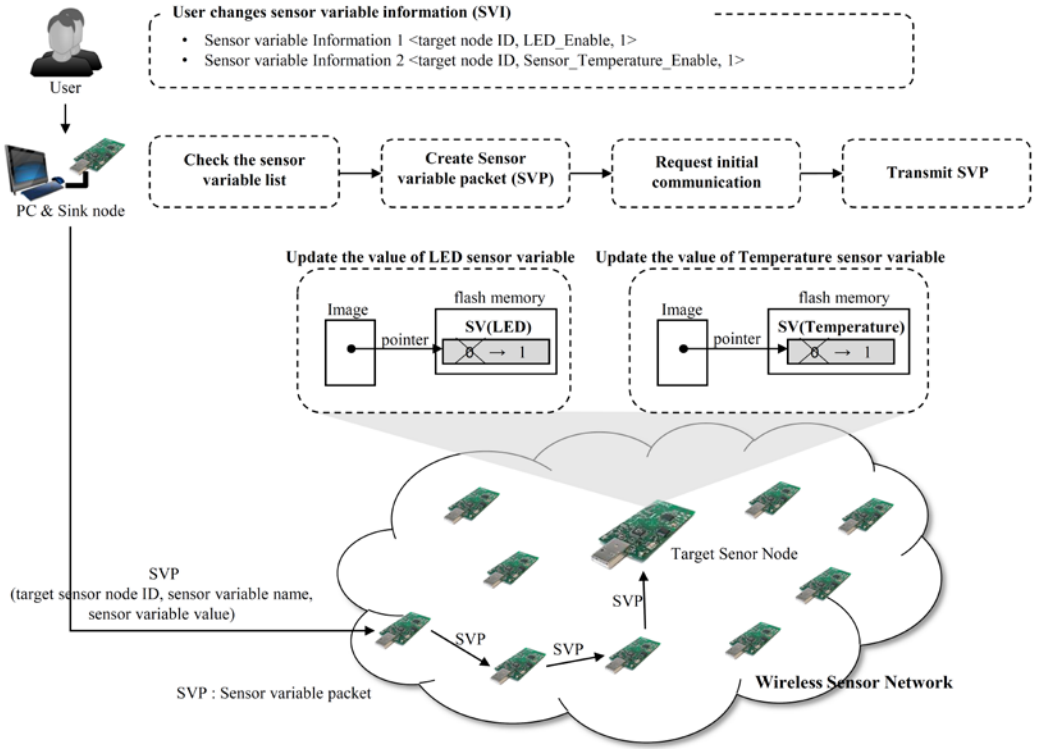

Fig. 9. Proposed reconfiguration method scenario

Fig. 10 shows the reconfiguration scenario execution. **Fig. 10**(a) shows the LED light on the sensor node is activated and blinking after the LED_Enable sensor variable is updated**. Fig. 10**(b) shows the On-state of the temperature sensor through the updating of the Sensor_Temperature_Enable sensor variable. The oscilloscope tool shows the temperature values sensed by the temperature sensor. The oscilloscope snapshot shows the values collected after the corresponding sensor variable change.

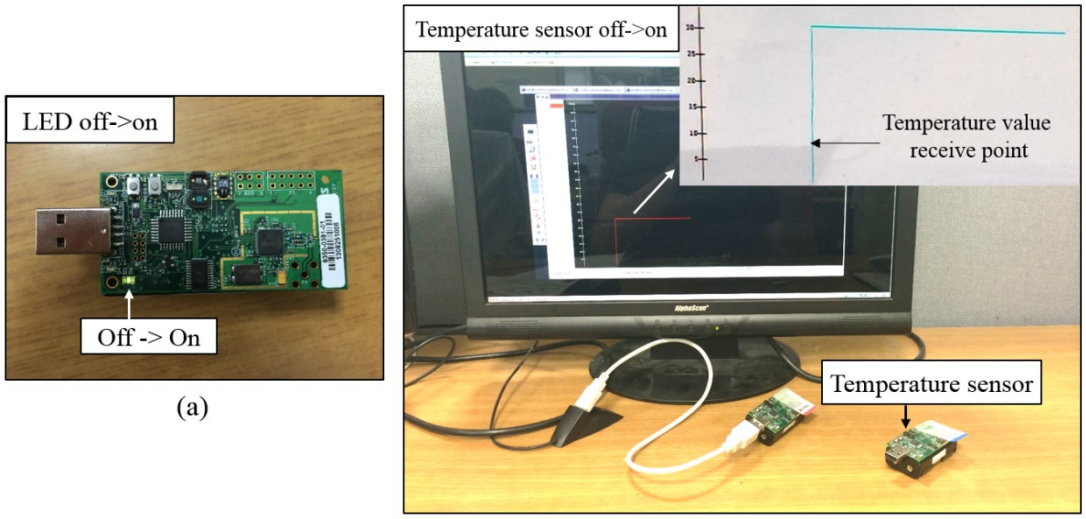

Fig. 10. Prototype of the sensor variable-based reconfiguration method

5. Experimental results

In this section, we present experimental evidence that our proposed method is superior to two other existing methods: (1) a full-image-based method and (2) a component-based method. For the experiments, Deluge and RemoWare were selected from among several full-image-based and component-based methods, respectively. In most cases, sensor networks suffer considerably from the network congestion problem and rapid consumption of the sensor node batteries. Therefore, the experiments were divided into two different types, as follows:

- **Network congestion:** The objective of this set of experiments was to study the effect of each method on network congestion in a sensor network. If the network congestion of the sensor network is mitigated to a greater degree when the proposed method is used than when the other methods are used, the proposed method is determined to be useful for building intelligent sensor networks.
- **Battery consumption:** In this set of experiments, the extent of the battery consumption of sensor nodes comprising a sensor network was compared. To determine the best method, we needed to identify the sensor node batteries that were consumed at the lowest rate when each different method was used.

5.1 Network Congestion

Network congestion frequently occurs when sensor nodes carry an excessive amount of data and leads to deterioration in an application's QoS, since the throughput of the sensor network is rapidly diminished by the network congestion problem. In our evaluation of Deluge[3] and our proposed method in terms of network congestion, to quantitatively measure the extent of the network congestion, we considered both the network bandwidth and the updated time. The

[3] In the experiment, our method is directly compared to not Remoware but Deluge because our proposed method and Deluge are running on the same operating system (TinyOs).

network bandwidth constitutes the channel capacity of a logical communication path in a sensor network. For instance, to transmit 10 kilo bytes in a file to a target sensor node n_2 from a sink node n_1, first a channel is set up between n_1 and n_2. Assuming that the channel has a capacity of 512 bytes per second, it will take about 20 s for n_2 to receive the entire file. However, if the file is only 1 kilobytes in size, it will take approximately 2 s. In addition, the update time is defined as the total amount of time it takes to send a file from n_1 to n_2 and then to install and render it ready for use in n_2.

Fig. 11 shows the experimental results about the total size of data (S) to be transmitted to a target sensor node (n_2) from the sink node (n_1) using Deluge and our approach. Formally, the total size of data sent by the Deluge and our approach can be computed by the following equations:

$$S_{Deluge} = S_{Os} + S_{Application}, \text{ and}$$
$$S_{Proposed\_Method} = S_{Sensor\_Node\_ID} + \sum_i (S_{Reconfigurable\_Sensor\_Variable\_Name} + S_{Value})$$

, where i means the i-th pair of sensor variable name and value. S_{Deluge} is composed of two terms, S_{Os} and $S_{Application}$, for reconfiguration. S_{Os} is the size of operating system for n_2, such as TinyOS and contiki, and $S_{Application}$ is the size of the re-compiled version for n_2. Deluge always sends to n_2 a ROM file including both S_{Os} and $S_{Application}$. On the other hand, $S_{Proposed\ method}$ is only a few bytes for sending a sensor ID followed by pairs of reconfigurable sensor variable names and values. In Deluge, suppose that the size of the entire operation system is about 10KB, and the size of the application is 5KB or so. S_{Deluge} is 15 KB. In our approach, if there are about 256 sensor nodes and variables in a wireless sensor network, each of $S_{Sensor\_Node\_ID}$, $S_{reconfigurable\_Sensor\_Variable\_Name}$, and S_{value} is 1 byte. If 12 pairs of reconfigurable sensor variable names and values, $\sum_i (S_{Reconfigurable\_Sensor\_Variable\_Name} + S_{Value})$ is 24 bytes. In this set-up, S_{Deluge} is about 600 times larger than $S_{Proposed\_Method}$. This outcome is similar to our empirical results in **Fig. 11**.

For the experiments, we prepared three test cases. One case constituted updating the configuration value of the blinking light period in a target sensor node, the second constituted updating the configuration value of the LED blinking On/Off state in a target sensor node, and the third constituted updating the configuration value of the temperature On/Off state in a target sensor node. For clarification, we hereafter call the three cases Blinking_Period, LED_Enable, and Sensor_Temperature, respectively. We also used 12 TelosB nodes running on TinyOS. We modified Deluge and implemented our method using benchmark data sets given by TinyOS. In our approach, the configuration information was updated on the flash memories of the TelosB nodes.

In all three test cases, when we used Deluge, the total data size was about 20 kilobytes on average, while data of about 25 bytes needed to be sent to n_2 when our method was used. The data size to be sent using Deluge is thus about 800 times bigger than that using our method. Such relatively large data cause or exacerbate the network congestion problem in sensor networks. In Deluge, the binary code of all the source codes, including OS and application software, should be sent to a target sensor node. In contrast, in our scheme, the only data to be sent contain the identifier of a target sensor node, reconfigurable sensor variable name, and reconfiguration value. If there are about 512 sensor nodes in a given sensor network, only 9 bits are required to identify each of all the sensor nodes. In addition, suppose that the total number of reconfigurable sensor variables is 64. In this case, only 6 bits are needed to identify each of the sensor variables. In most cases, the reconfiguration information of sensor variables constitutes, for example, turning a particular sensor on/off and changing the sensing period.

Thus, 1 byte is sufficient to represent the reconfiguration value. As a result, when our approach is used, the total size of data to be transmitted to n_2 is at most 3 bytes. In Deluge, if the capacity of the channel between n_1 and n_2 is 512 bytes, n_1 has to divide the total data into 20 parts of data, and then, each part of the data is sent to n_2. Therefore, the network bandwidth is occupied and cannot be used by other nodes until n_2 has received all the data. In contrast, when our method is applied to the sensor network, all the data are simultaneously sent to n_2 and the used channel is quickly available. Based on these experimental results, we suggest that our approach outperforms the existing methods such as Deluge in the aspect of network bandwidth. Therefore, using our approach ameliorates the network congestion problem that frequently occurs in sensor networks.

We also measured the updated time (T) of Deluge and our method. Regardless of the Deluge or our approach, the updated time for the reconfiguration is defined as

$$T_{Update} = \sum_j (S_{File\_Size} / S_{Bandwidth}) + T_{Install\_Time}$$

, where j means the distance to the target sensor node from the sink node. The communication between sensor nodes is made by UDP protocol. The sink node first sends a segment to the nearest neighbor of the sensor nodes in a wireless sensor network. The neighbor node stores and forwards it to its nearest neighbor again. In this way, the segment is arrived to the target sensor node. The UDP protocol is connectionless without providing any control scheme for end-to-end network flow. Thus, the updated time can be defined as the combination of the data transmission time and the install time. The data transmission time is computed by S_{File_Size} / $S_{Bandwidth}$. For example, S_{Deluge} / $S_{Bandwidth}$ in the Deluge method, but $S_{Proposed\_Method}$ / $S_{Bandwidth}$ in our method. Please note that the transmission time is the dominant term in T_{Update} equation. Compared to our approach, the size of Deluge is considerably larger than that of our method. This indicates that the T_{Update} time of Deluge is also much slower than ours. In addition, another term, $T_{Install\_Time}$ is the amount of time taken to install the binary image in the target sensor node using the Deluge method. However, in our method, it means the amount of time taken to simply update the values of the sensor variables. In Deluge, if S_{Deluge} is about 15KB, and $S_{Bandwidth}$ is 250 kbps (please, refer to TelosB in http://www.memsic.com), then it takes about 60 seconds to send S_{Deluge} to the target sensor node. Further, if the distance from the sink node to the target sensor node is 5, then it takes about 65 seconds by $\sum_j (S_{File\_Size} / S_{Bandwidth})$. Supposing that $T_{Install\_Time}$ is about 3 seconds, the T_{Update} time is 68 seconds. In case of our method, if $S_{Proposed\_Method}$ is 25KB, and $S_{Bandwidth}$ is 250 kbps, it takes about 1 second to send to the target sensor node by $S_{Proposed\_Method}$ / $S_{Bandwidth}$. In addition, if the distance is 5, it takes about 2 seconds by $\sum_j (S_{File\_Size} / S_{Bandwidth})$. If $T_{Install\_Time}$ is 1 second, the T_{Update} time is 3 seconds. As a result, the T_{Update} time of the proposed method is about 23 times faster than that of Deluge. Figure 11 also shows the similar patterns between the analytical and empirical studies.

In the set of experiments, the average update time of Deluge was more than 1 m. In contrast, the update time required by the proposed method was at most 2 s on average. The experimental results show that our proposed method is about 30 times faster than existing full-image-based methods such as Deluge. Interestingly, the update time consists of transmit time and installation time. The transmit time is defined as the amount of time it takes to send all data to a target sensor node from a sink node. The install time is the amount of time it takes to install the data in the target sensor node. In Deluge, the transmit time and the installation time were about 61 s and 4 s, respectively. That is, the transmit time is dominant in the update time. Similarly, in our approach, the transmit time is about 2 s, whereas the installation time is at

most 1 ms. A comparison of the transmit time of our scheme and of Deluge reveals that the proposed method is also about 30 times faster than Deluge. This result is consistent with the experimental results for the network bandwidth.

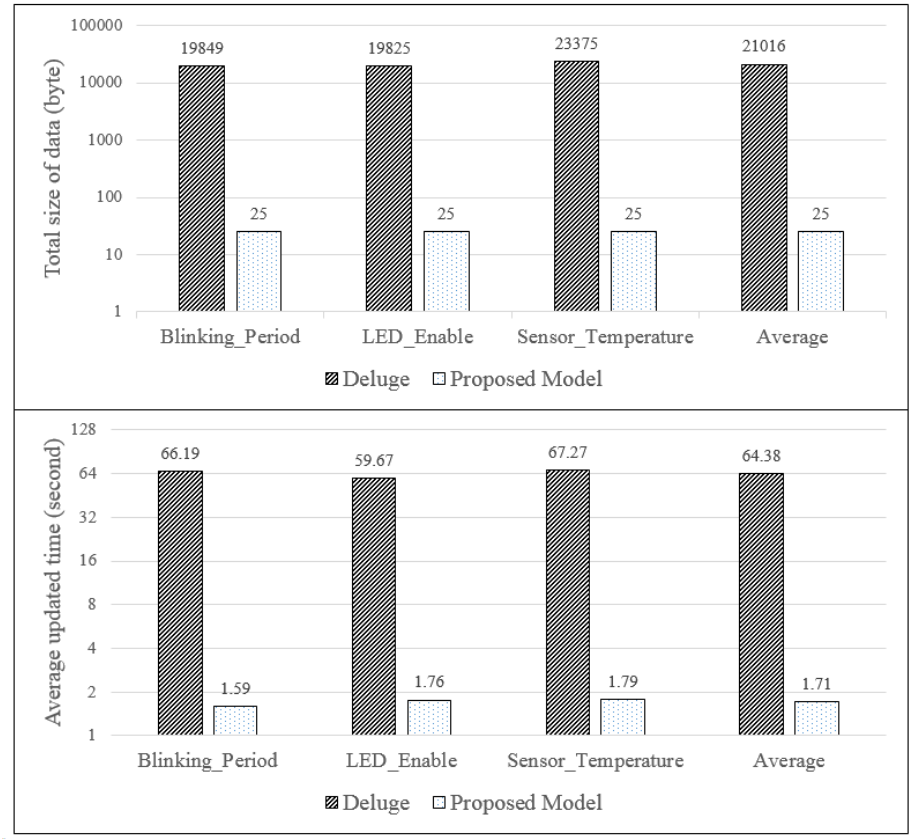

Fig. 11. Total size of data and average updated time results

5.2 Battery Consumption

By referring to IEEE 802.15.4 multihop networks [33], we implemented a simulator for estimating the energy consumption of Deluge, RemoWare, and our approach. In the simulation, we focused on updating the configuration value of the LED blinking On/Off state in a target sensor node. In addition, we used OMNet++ [34], a well-known component-based C++ discrete even simulator jointly with the MiXiM and OMNet++ frameworks created for various types of mobile and fixed wireless networks (e.g., WSNs, body area networks, ad-hoc networks, and vehicular networks) [35]. The simulated network nodes were located in a uniformly random spatial distribution in two-dimensional square areas. The nodes were configured to use the 2.5 GHz band IEEE 802.15.4 physical layer and parameters in CC2420 [36]. We conducted experiments and activated the parameter settings of the simulator on the basis of experimental results and similar simulation environments [37]. Thirty sensor nodes were randomly deployed in a 500 m * 500 m field.

Fig. 12 shows a screen-shot image of our simulation based on OMNeT++. In the figure, node[0] represents a sink node and the remaining nodes are sensor nodes. Each sensor node has a borderline of a circle-shape within which it can only communicate with other sensor

nodes. The arrows between the nodes represent the routing information. Data are transmitted to sensor nodes based on the routing information.

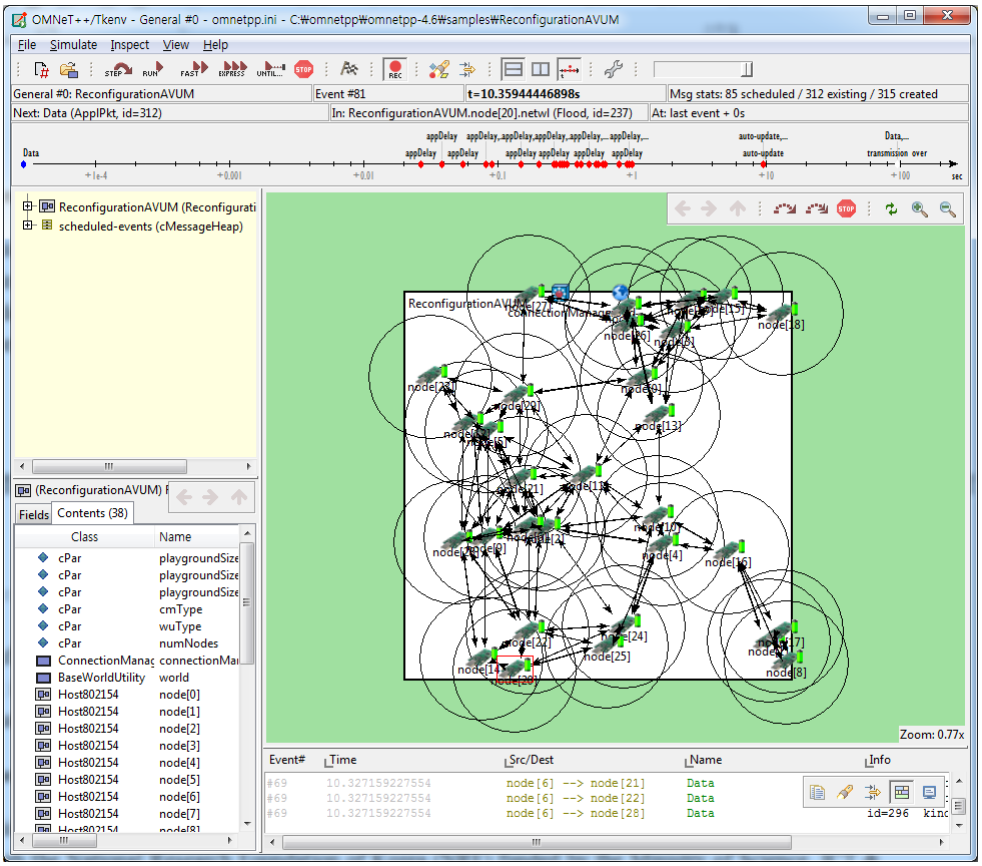

Fig. 12. Screen-shot image of our simulation based on OMNeT++

Fig. 13 illustrates the experimental results for the battery consumption of sensor nodes when each method, Deluge, RemoWare, or our approach, was employed. In the figure, the x-axis represents each of the 30 sensor node identifiers and the y-axis represents the battery consumption per sensor node. The average energy consumptions of Deluge, RemoWare, and our approach are 108.5 mWs, 47.4 mWs, and 6.1 mWs, respectively. According to these experimental results, our proposed method is 17 times and 7 times more energy-efficient than Deluge and RemoWare, respectively. This implies that our method can save battery consumption efficiently, as compared with the existing methods. The reason is that only data of 3 bytes are sent to a target sensor node in our approach and therefore the target sensor node tends to consume less energy because it is handling a small size of data. In contrast, the other methods are likely to require large batteries to handle the relatively large data. For instance, each sensor node has a receiver and a transmitter. The receiver and transmitter consume a battery energy of 0.6391 mA and 0.6845 mA per (receive and transmit) operation. In the example in the previous section, when we used Deluge, the receiver of a target sensor node (n_2) had to be activated 20 times to receive all the data. Then, it needed to consume 20 × 0.6391 mA = 12.782 mA to receive every data item. In contrast, in our approach, the receiver can be activated only one time and therefore it consumes only 0.6391 mA.

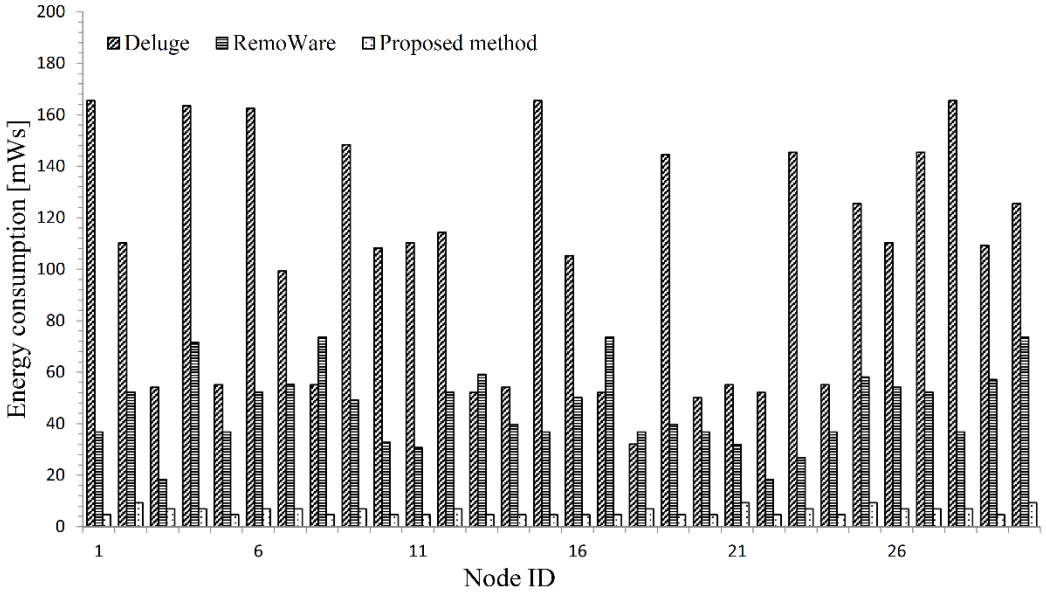

Fig. 13. Experimental results for battery consumption

6. Conclusion

In this paper, we proposed a novel approach for resolving the reconfiguration issue that is a significant challenging problem in sensor networks and the IoT ecosystem. Whenever configuration values are updated in a sink node (e.g., a certain sensor is turned on or off), the updated information should also be updated in the target sensor nodes. In existing methods, such as full-image-based (e.g., Deluge) and component-based (e.g., RemoWare) methods, the binary code of either the entire or part of the source code should be sent to the target sensor nodes, although only the configuration information in the entire source code is changed in the sink node. For this reason, the existing methods frequently result in both the network congestion problem and battery requirement issues, which are very important in sensor networks. In contrast, since our approach is based on updating only configuration information, it can ease the limitations of the existing methods.

In future work, we plan to study the autonomous reconfiguration problem. To address this problem without human intervention, configuration information should be automatically updated in the target sensor nodes to provide flexibility in handling a situation.

Acknowledgement

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2014R1A1A2058992). The co-corresponding authors are Doo-Kwon Baik and Dongwon Jeong.

References

[1] Vermesan, O., Friess, P. "Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems," *River Publishers: Aalborg 42 9000 Aalborg*, Denmark, 2013. Article (CrossRef Link).

[2] Atzori, L., Iera, A., Morabito, G., "The Internet of Things: A survey," *Comput. Network*, 10, 2787-2805, 2010. Article (CrossRef Link).

[3] Hui, J.W., Culler, D. "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proc. of the 2nd International Conference on Embedded Networked Sensor Systems*, 2004. Article (CrossRef Link).

[4] Marron, P.J., Gauger, M., Lachenmann, A., Minder, D., Saukh, O., Rothermel, K. "Flexcup: A flexible and efficient code update mechanism for sensor networks," in *Proc. of the 3rd European Conference on Wireless Sensor Networks*, 3868, 212-227, 2006. Article (CrossRef Link).

[5] Coulson, G., Blair, G., Grace, P., Taiani, F., Joolia, A., Lee, K., Ueyama, J., Sivaharan, T., "A generic component model for building systems software," *ACM Transactions on Computer Systems*, 26(1), 1-42, 2008. Article (CrossRef Link).

[6] Grace, P., Coulson, G., Blair, G., Porter, B., Hughes, D. "Dynamic reconfiguration in sensor middleware," in *Proc. of the International Workshop on Middleware for Sensor Networks*, MidSens '06, 1-6, 2006. Article (CrossRef Link).

[7] Mottola, L., Picco, G.P., Sheikh, A.A. "FiGaRo: Fine-grained software reconfiguration for wireless sensor networks," *Wireless Sensor Networks Lecture Notes in Comput. Sci.*, 4913, 296-304, 2008. Article (CrossRef Link).

[8] Hughes, D., Thoelen, K., Horre, W., Matthys, N., Cid, J.D., Michiels, S., Huygens, C., Joosen, W. "LooCI: A loosely-coupled component infrastructure for networked embedded systems," in *Proc. of the 7th International Conference on Advances in Mobile Computing and Multimedia*, MoMM '09, 195-203, 2009. Article (CrossRef Link).

[9] Taherkordi, A., Loiret, F., Rouvoy, R., Eliassen, F. "A generic component-based approach for programming, composing and tuning sensor software," *Comput. J.*, 54, 1248-1266, 2011. Article (CrossRef Link).

[10] Dunkels, A., Gronvall, B., Voigh, T., "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Proc. of 29th Annual IEEE International Conference on Local Computer Networks*, 455-462, 2004. Article (CrossRef Link).

[11] Szczodrak, M., Gnawali, O., Carloni L.P. "Dynamic reconfiguration of wireless sensor networks to support heterogeneous applications," in *Proc. of IEEE DCOSS Conference*, 38, 23-31, 2005. Article (CrossRef Link).

[12] Leligou, H.C., Redondo, L., Zahariadis, T., Retamosa, D.R., Karkazis, P., Papaefstathiou, I., Voliotis, S., "Reconfiguration in wireless sensor networks," *Developments in E-systems Engineering*, 59-63, 2010. Article (CrossRef Link).

[13] Hughes, D., Canete, E., Daniels, W., R, G.S., Meneghello, J., Matthys, N., Maerien, J., Michiels, S., Huygens, C., Joosen, W., Wijnants, M., Lammtte, W., Hulsmans, E., Lannoo, B., Moerman, I. "Energy aware software evolution for wireless sensor networks," in *Proc. of IEEE 14th International Symposium and Workshops on World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 1-9, 2013. Article (CrossRef Link).

[14] Yeh, C.T. "Dynamic reconfiguration techniques for wireless sensor networks," *University of Massachusetts Amherst Masters Thesis*, 2008. Article (CrossRef Link).

[15] Cetina, C., Giner, P., Fons, J., Pelechano, V., "Autonomic computing through reuse of variability models at runtime: The case of smart homes," *IEEE Computer 2009*, 42(10), 37-43, 2009. Article (CrossRef Link).

[16] Ortiz, Ó., García, A. B., Capilla R., Bosch J., Hinchey M., "Runtime variability for dynamic reconfiguration in wireless sensor network product lines,' in *Proc. of the 16th International Software Product Line Conference*, 2, 143-150, 2012. Article (CrossRef Link).

[17] Mouronte, M. L., Ortiz, Ó., García, A. B., Capilla R., "Using dynamic software variability to manage wireless sensor and actuator networks," *IM 2013*, 1171-1174, 2013. Article (CrossRef Link).
[18] Gámez, N., Fuentes L., "FamiWare: a family of event-based middleware for ambient intelligence," *Personal and Ubiquitous Computing*, 15(4), 329-339, 2011. Article (CrossRef Link).
[19] Delicato, F. C., Fuentes, L., Gámez N., Pires P. F., "Variabilities of wireless and actuators sensor network middleware for ambient assisted living," *IWANN 2009*, 851-858, 2009. Article (CrossRef Link).
[20] Jung, E.H., Kim, Y.P., Park, Y.J., Cho, S.Y., Han, S.Y., "An attribute-based naming architecture for wireless sensor networks using a virtual counterpart overlay network," *EUROSSC 2006*, LNCS 4272, 222-225, 2006. Article (CrossRef Link).
[21] Jung, E.H., Kim, Y.P., Park, Y.J., "TinyONet: A cache-based sensor network bridge enabling sensing data reusability and customized wireless sensor network services," *Sensors*, 8, 7930-7950, 2008. Article (CrossRef Link).
[22] Zhang, J., Ren, F., He, T., Lin, C., "Attribute-aware data aggregation using dynamic routing in wireless sensor networks," in *Proc. of IEEE International Symposium on World of Wireless Mobile and Multimedia Networks*, 1-9, 2010. Article (CrossRef Link).
[23] Liu, X., Li, J., Kang, G., "A Smart Energy-efficiency Deployment Scheme for Lifetime Enhancement in Large-scale Wireless Sensor Networks," *Smart Computing Review*, 5, 591-601, 2015. Article (CrossRef Link).
[24] Bakshi, A., Prasanna, V.K., Reich, J., Larner, D., "The abstract task graph: A methodology for architecture-independent programming of networked sensor systems," in *Proc. of the 2005 Workshop End-to-end, Sense-and-Respond Systems, Applications and Services*, 19-24, 2005. Article (CrossRef Link).
[25] Pathak, A., Mottola, L., Bakshi, A., Prasanna, V.K., Picco, G.P., "Expressing Sensor Network Interaction Patterns using Data-Driven Macroprogramming," in *Proc. of the 5th IEEE International Conference on Pervasive Computing and Communications Workshops*, 255-260, 2007. Article (CrossRef Link).
[26] Pathak, A., Mottola, L., Bakshi, A., Prasanna, V.K., Picco, G.P, "A compilation framework for macroprogramming networked sensors," in *Proc. of the 3rd International Conference on Distributed Computing in Sensor Systems*, 189-204, 2007. Article (CrossRef Link).
[27] Lee, W.J., Kim, J.I., Kang, J.M. "Automated construction of node software using attributes in a ubiquitous sensor network environment," *Sensors*, 10, 8663-8682, 2010. Article (CrossRef Link).
[28] Jung, H.J., Jeong, D.W., Lee, S.H., "Self-reconfiguration middleware model for sensor network environment," *Research Notes in Information Science (RNIS)*, 14, 2013. Article (CrossRef Link).
[29] Jung, H.J., Jeong, D.W., Lee, S.H., "Self-reconfiguration middleware model and qualitative evaluation for sensor network environment," *International Journal of Advancements in Computing Technology (IJACT)*, 5, 174-183, 2013. Article (CrossRef Link).
[30] Attribute-based programming. Available online: http://www.webopedia.com/TERM/A/attribute_based_programming.html (accessed on 21 January 2015).
[31] Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., et al., "TinyOS: An operating system for sensor networks," *Ambient Intelligence*, Springer Berlin Heidelberg: Berlin, Germany, pp. 115-148, 2005. Article (CrossRef Link).
[32] TELOSB MOTE PLATFORM. Available online: http://www.willow.co.uk/TelosB_Datasheet.pdf (accessed on 21 January 2015).
[33] IEEE 802.15.4. (2006). Wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (WPANs)
[34] Varga, A. "The Omnet++ discrete event simulation systems," in *Proc. of the European Simulation Multiconference*, Prague, Czech Republic, 6-9 June 2001.
[35] MiXiM simulator for wireless and mobile networks using OMNeT++. Available online: http://mixim.sourceforge.net/ (accessed on 21 January 2015).

[36] Texas Instruments Incorporated. 2.4 GHz IEEE 802.15.4/ZigBee-ready RF Transceiver. Available online: http://www.ti.com/lit/gpn/cc2420 (accessed on 1 January 2015).
[37] Kermajani, H., Gomez, C., "On the network convergence process in RPL over IEEE 802.15.4 multihop networks: Improvement and trade-offs," *Sensors*, 14, 11993-12022, 2014. Article (CrossRef Link).

Hyunjun Jung received a M.S. degree in the Department of Computer Science and Engineering from Soongsil University. Seoul, Republic of Korea in 2010. He is candidates of Ph.D. degree in the department of computer science and engineering, Korea University, Seoul, Korea. His interested in Data engineering, the Semantic Web, Ubiquitous computing, Big data, Software engineering.

Dongwon Jeong received his Ph.D. degree in Computer Science from Korea University, Korea, 2004. He was a Research Assistant Professor, Korea University, 2004-2005 and was a Visiting Research Scholar (PostDoc.), School of Information Sciences & Technology, Pennsylvania State University, USA, 2005. He was a Visiting Research Scholar, Department of Computer Science and Engineering, Oakland University, USA, 2013-2014. He is a Professor in Dept. of Statistics & Computer Science, Kunsan National University, Korea from 2005. His research interests include data engineering, the semantic web, smart mobile services, dig data, IoT, and Security.

Byung-won On received the PhD degree from the Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, in 2007. He is an assistant professor in the Department of Statistics and Computer Science, Kunsan National University, Gunsan-si, Jeollabuk-do, Korea. His research interests are probabilistic models, entity resolution and search, social network analysis and mining, PCM-based in-memory databases, and cutting-edge big data technologies.

Doo-Kwon Baik received his M.S. and Ph.D. degrees in Computer Science from Wayne State University in the U.S. in 1983 and 1986, respectively. He is a full professor in the Department of Computer Science and Engineering, Korea University, Seoul, Republic of Korea. He is interested in semantic web, ontology, data engineering, modeling & simulation, and metadata registry.