Event-Based Middleware for Healthcare Applications

Rossi Kamal, Nguyen H. Tran, and Choong Seon Hong

Abstract: **In existing middleware for body sensor networks, energy limitations, hardware heterogeneity, increases in node temperature, and the absence of software reusability are major problems. In this paper, we propose an event-based grid middleware component that solves these problems using distributed resources in in vivo sensor nodes. In our multi-hop communication, we use a lightweight rendezvous routing algorithm in a publish/subscribe system of event-based communication. To facilitate software reuse and application development, a modified open services gateway initiative has been implemented in our middleware architecture. We evaluated our grid middleware in a cancer treatment scenario with combined hyperthermia, chemotherapy, and radiotherapy procedures, using in vivo sensors.**

Index Terms: **Cancer treatment, healthcare, in vivo sensors, middleware.**

I. INTRODUCTION

With the evolution of modern diagnostic systems, in vivo sensors have been used to improve healthcare [1], [2].

Hyperthermia [3] is a prominent method for cancer treatment. In hyperthermia, body cells are heated to a certain temperature for a certain amount of time. Furthermore, radiotherapy and chemotherapy are popular cancer treatment methods, and the combined use of hyperthermia, radiotherapy, and chemotherapy [3] is considered the best approach to cancer treatment. However, human cells are very sensitive, and if the hyperthermia temperature exceeds a threshold value, the cells could be damaged, making temperature scheduling a critical issue in this approach.

Thermal aware routing algorithms [4], [5] have been proposed to solve the temperature scheduling problem using in vivo sensor nodes. However, the main problem is the necessary complexity, which cannot be achieved with lightweight in vivo sensor nodes. As the temperature increases, more heat is generated, and more battery is wasted. Thus, a lightweight routing algorithm for in vivo sensor nodes is needed.

Middleware [6], [7] for body sensor nodes [8], [9] has been increasing in popularity [10], [11], but middleware for in vivo sensor nodes has not yet been investigated in detail. In cancer treatments where heterogeneous in vivo sensor nodes are deployed for hyperthermia, radiotherapy, and chemotherapy, middleware may be a relatively good option. The miniature nodes must be lightweight and thus must distribute operation, temperature, and power consumption. This has motivated us to develop a grid approach [12] for in vivo sensor nodes.

We have proposed a grid middleware component that uses event-based communication [13], [14] in in vivo sensor nodes and an open services gateway initiative (OSGI)-based architecture [15]. The developed system consists of a lightweight rendezvous routing algorithm that schedules the temperatures for the in vivo sensor nodes during combined hyperthermia, radiotherapy, and chemotherapy procedures.

This paper is organized as follows. Section II discusses the need for grid-based middleware for in vivo sensor nodes. In Section III, we propose an OSGI-based [15] middleware architecture and a lightweight rendezvous algorithm. Section IV presents performance evaluations with our problem scenario. Section V discusses related works, and finally, Section VI discusses future directions.

Manuscript received September 30, 2010; approved for publication by Nei Kato, Guest Editor, April 18, 2012.

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2011-0020518).

The authors are with the Department of Computer Engineering, Kyung Hee University, Korea, email: {rossi, nguyenth, cshong}@networking.khu.ac.kr and C. S. Hong is the corresponding author.

II. THE NEED FOR GRID MIDDLEWARE FOR IN VIVO SENSOR NODES

In vivo sensor nodes are made of heterogeneous hardware components and are deployed for critical, sensitive, and complex applications. With improvements in technology, smart in vivo sensors that provide different services are becoming popular in various treatments. Middleware hides the underlying operating system and the heterogeneity of the physical devices from the application layer. However, with in vivo sensor nodes, the reduction of both energy dissipation and thermal effects is a major concern. Thus, grid middleware can be a good solution because it distributes services among the in vivo sensors and reduces both these quantities. Grid middleware can be beneficial for in vivo sensors in the following ways.

A. Scalability

Scalability is a system's capability of supporting a large number of in vivo sensor nodes. Middleware services are distributed; there are almost no centralized approaches, and decisions and state information are viewed locally, rather than globally. In addition, network bandwidth and memory are used efficiently.

B. Interoperability

Grid middleware is designed to support the heterogeneity of the components of the network and is language- and platform-independent. It can cope with the dynamic environment of the components of distributed systems, and both fixed and mobile devices can join and leave the distributed systems at run time.

C. Reliability

Quality of service (QoS) and reliability requirements can be resolved with grid architecture. Different in vivo sensors may have different requirements, while grid middleware may give

generic support to these requirements. For example, a generic media access control (MAC) framework can be integrated into the middleware architecture. In vivo sensor nodes can use any of the MAC protocols depending on their specific criteria.

III. PROPOSED MIDDLEWARE

A. Motivation

Due to its service-oriented design, OSGI has become ideal for grid computing. The problems in Jini and UPnP have set the stage for OSGI-based design of grid middleware for body sensor networks (BSNs). The limitations of Jini include a lack of platform independency in remote method invocation (RMI)-based implementations and the management overhead for the central look-up server, while the disadvantages of UPnP include type mapping overhead, the high verbosity of extensible markup language (XML), and a decreased performance with the simple object access protocol (SOAP). However, for any service, loosely coupled components of different OSGI instances on mobile middleware of different sensor nodes [16] can interact with each other, as in a grid environment.

Bundles and services are major components of the OSGI framework. A bundle is a package archive containing software written for the OSGI framework. It represents a functional component that can be installed, activated, deactivated, and uninstalled. Bundles can import or export packages from or to other bundles and can provide services and register them with the framework's service repository. A service is an object that has well-defined functionalities defined by object implementations; it consists of a collection of interfaces and their different implementations. A service is registered by a bundle to the framework and can thus be recognized by other bundles. There is a close relation between bundles and service in the OSGI specification. A bundle may register any number of services. Because services are interfaces, they may have more implementations, and one implementation may belong to different bundles. For example, a service could be wireless communication, wherein implementations may include Bluetooth or Wi-Fi.

In our middleware, the distributed features of the local OSGI framework are masked, and remote services are accessible as if they were present in the framework. OSGI transparency is ensured as well. Although the Java virtual machine (JVM) is absent from this OSGI architecture, there are no other restrictions on the OSGI specification, and all existing bundles are reusable; therefore, OSGI noninvasiveness is maintained. This architecture shows that a remote node for a service is the same as that of the local framework, thereby allowing consistency to be maintained. This architecture allows the OSGI framework to operate on a large range of body sensor nodes, and distribution does not limit the configurations in which OSGI can be used; thus, generality is ensured. This system also prevents a BSN node from being overwhelmed by the number of available services, and the "statement of supply and demand" is ensured.

Grid middleware for BSN should be lightweight owing to the energy constraints of body sensors. Existing sensor nodes utilize component programming of nesC over TinyOS, the most popular operating system for BSN. To design such lightweight middleware, we must adopt this development environment. The OSGI design paradigm uses JVM over TinyOS. In this middleware, we use the OSGI paradigm with nesC over TinyOS. Thus, OSGI instances on different body sensor nodes use the component programming of TinyOS to communicate within the grid environment.

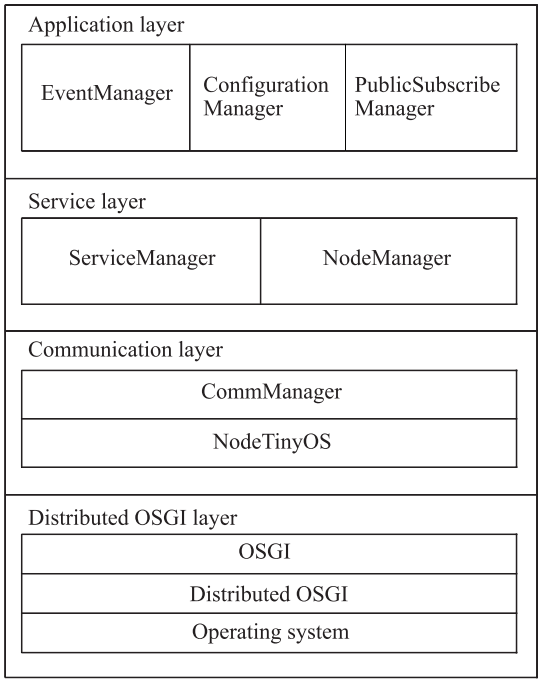

Fig. 1. Architecture of the proposed middleware.

B. Architecture

The proposed grid middleware (Fig. 1) consists of several layers.

B.1 Application Layer

The application layer is the highest level of the middleware. It consists of sub-layers EventManager, PublishSubscribeManager, and ConfigurationManager. Applications such as sensing, comparison, and temperature control are related to EventManager; publish-subscribe applications governed by the proposed lightweight rendezvous algorithm are related to PublishSubscribeManager; while applications involving the installation, reinstallation, and uninstallation of sensor devices are related to ConfigurationManager. All applications are considered as OSGI-bundled, and thus are installed in the application layer. The benefit of using OSGI in the middleware is the ease of installation from the user's point of view. When we need to add a new device to the system, an OSGI bundle implementing the device's service should be installed. Inside the middleware, ServiceManager uses the LoadNode method and passes the device bundle's service location and the serial identifier of the node. The installation process is performed in ServiceManager and is transparent to the end user, who only needs to enter the node's serial identifier; this can be done by simply clicking on a button labeled "load sensor node."

B.2 Service Layer

The service layer consists of two sub-layers: ServiceManager and NodeManager.

ServiceManager allows application bundles to use their services by providing the corresponding objects. The objects can be of either action, publish-subscribe, or configuration types. For example, to add a new node to the system, ServiceManager creates an object of ConfigurationManager. Corresponding service instances are created dynamically and are allocated to that application bundle. In this case, the service instances are address assignment, hardware configuration, and device status that updates with a beacon message transfer to every nodes. The LoadNode method of ServiceManager is involved in the entire process. ServiceManager is assigned to update each device status. It sends a beacon message to the other sensor nodes to update the status and forwards incoming messages to the application bundles. Application bundles process the message to retrieve service information.

NodeManager provides a service that allows sensor nodes to communicate with an 802.15.4 message. It is assisted by CommManager of the communication layer.

The service layer is the most important layer in the architecture. A set of open interfaces can be defined in this layer to solve the interoperability problem resulting from sensor devices and services from different manufacturers. Based on energy efficiency, different MAC protocols for the sensor network are available. A generic MAC platform can also be considered in the layer.

B.3 Communication Layer

CommManager in this layer receives messages and sends them to the upper layer. Message communication is performed through a serial port. CommManager, NodeTinyOS, and NodeManager are involved in the registration of different types of received messages. Message types can be subscribed, published, or brokered. Messages sent from other in vivo sensor nodes are received by NodeTinyOS. NodeManager receives a message using the CommManager service. The message is then sent to ServiceManager, which checks the message source address to forward it to the application bundle related to the sensor node information. This determines whether the message is from a publisher, subscriber, or broker node and the message is dealt with accordingly. The application bundle processes the incoming message and forwards it to other application bundles that will perform the required services.

B.4 Distributed OSGI Layer

The lowermost layer is the distributed OSGI layer, which allows communication among multiple peers within the OSGI framework, in a manner similar to a large OSGI framework. This layer allows distributed applications to be constructed with OSGI modularity. In this layer, OSGI bundles for different services are transparently distributed along module boundaries.

C. Functionality

The proposed middleware has a generic interface so that it can operate over underlying heterogeneous sensor nodes. We have described this middleware as a generic sensing service in Fig. 2. Sensor nodes of different sensing services, such as temperature, pressure, heartbeat, and oxygen or glucose calculators, from different manufacturers can be installed with this feature.

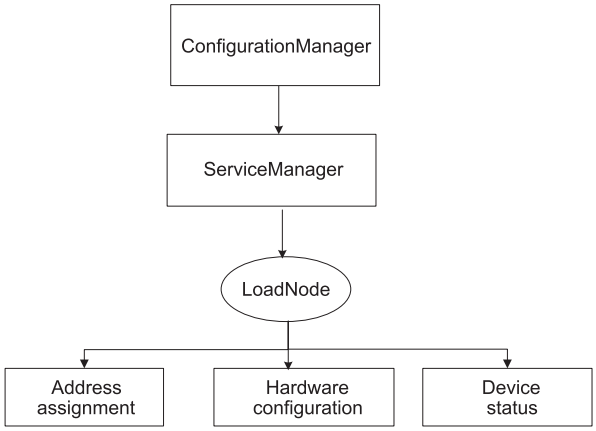

Fig. 2. Generic sensing service is provided by ConfigurationManager module in middleware. When to add a new sensor device or service, ServiceManager creates an object of ConfigurationManager with its instances like address manager, hardware configuration, and device status.

This generic framework supports different communication protocols, including 802.15.4 and Bluetooth. We have described this as generic communication in Fig. 3. The uniqueness of the middleware is the generic support for event-based communication. This provides a generic publish-subscribe mechanism that operates with any of the underlying sensor nodes and with both single and multiple sensing phenomena. The event interface provides the ability to define an event independent of the underlying sensing devices.

In our experiment, we considered a publish-subscribe mechanism for a temperature sensor node in an implanted sensor network. An event is the threshold heat generated in the implanted sensor network. However, this publish-subscribe mechanism is applicable to single and multiple sensing phenomena. Events can be chosen on the basis of specific requirements. For example, we can consider an implantable sensor network with blood oxygen calculations and temperature- and humidity-sensing abilities. We can apply our generic publish-subscribe mechanism to this network on the basis of events such as an energy dissipation threshold for these nodes.

C.1 Generic Sensing Service

The ConfigurationManager interface supports middleware over heterogeneous sensor devices and sensing services. Applications such as installation, reinstallation, and uninstallation of devices are performed by ConfigurationManager as OSGI bundles. All applications are considered as OSGI bundles in the middleware. For example, when we want to add a device to the system, an OSGI bundle is created with those device properties. Inside the middleware, ServiceManager creates an object of ConfigurationManager. Corresponding service instances such as address assignment, hardware configuration, and device status updating, are created immediately. ServiceManager then calls LoadNode by transmitting the OSGI bundle service loca-

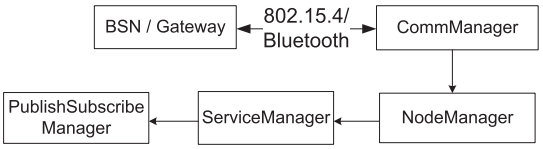

Fig. 3. Generic communication is provided by CommManager module in middleware. It allows middleware to work with other BSN or gateway node in 802.15.4 or Bluetooth protocol.

tion and node serial identifier. Next, LoadNode sends a beacon message to other nodes to update the status and forwards incoming messages to the OSGI bundle, which processes that message to retrieve service information.

C.2 Generic Communication

CommManager is a generic interface that provides the middleware with the ability to communicate with protocols (for example, Bluetooth or 802.15.4). NodeTinyOS receives a message from other sensor nodes. NodeManager then receives the message using CommManager. Next, the message is sent to ServiceManager, which checks message properties and forwards it to the appropriate application bundle based on the message type. This application bundle processes the incoming message and forwards it to other application bundles that will perform the required services.

C.3 Generic Publish-Subscribe with Fault-Tolerancy

Fault tolerance is necessary in healthcare applications. A single point of failure can be crucial in a healthcare application with BSNs. The generic publish-subscribe mechanism of our middleware supports fault tolerance among sensor nodes and the rendezvous node.

Fig. 4 shows the generic publish-subscribe mechanism with fault tolerance. Temperature generated in body sensor nodes can be assumed beforehand. Based on this, we divide all body sensor nodes into small clusters and assign a subscriber, publisher, or broker role to each node in a cluster. This information is installed in a rendezvous node. The first rendezvous node notifies every other node of the cluster information and role (steps 1–3). In each cluster, a subscriber node subscribes to a broker node for a specific event (step 4). The broker node then sends all successful subscription information to the rendezvous node (step 5). The subscription information is a list of successfully subscribed nodes and corresponding events. Then, the rendezvous node verifies the corresponding cluster information and checks whether every subscriber's subscription information has been received. Data loss might occur either between the subscriber and broker or between the broker and rendezvous nodes. If any subscriber's subscription information is missing, the rendezvous node waits for the corresponding event to occur. The rendezvous node learns of the corresponding event occurrence from the notification confirmation message of a lightweight publisher (steps 6–10) and directly communicates with both the publisher and the subscriber to immediately stop and start their services (steps 10–13). Generally, when an event occurs, the publisher immediately stops the related service and sends a notification message to the broker (step 6),

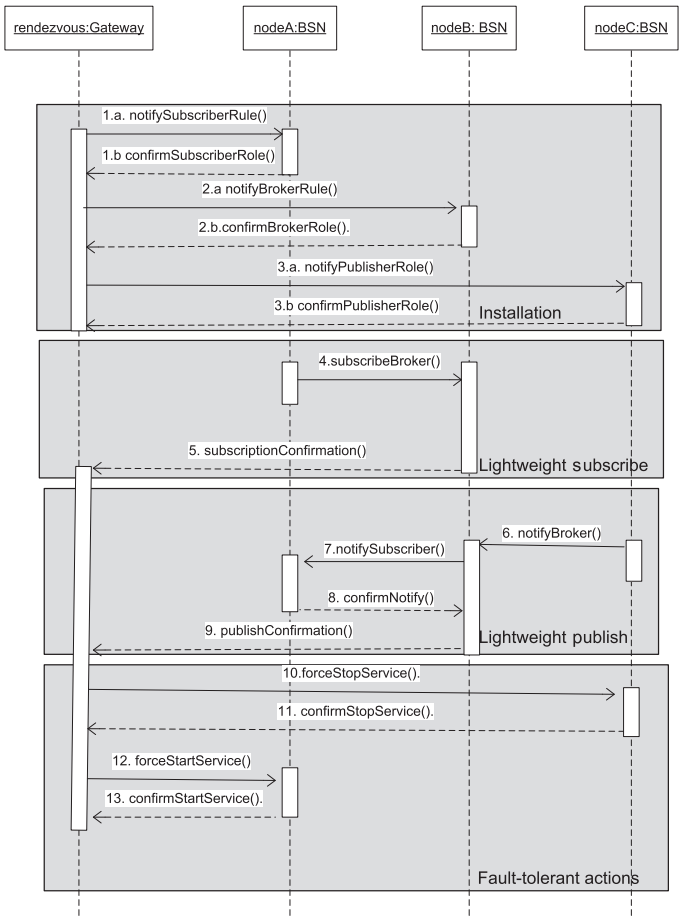

Fig. 4. Sequence diagram of the fault tolerant functionality of proposed middleware on rendezvous node, body sensor nodes (node A, node B, node C of the same cluster). Rendezvous node sends subscriber, broker and publisher role to nodes A, B, and C, respectively (steps 1–3). Node A (subscriber) is subscribed to node B (broker) in lightweight subscription (steps 4 and 5). When an event occurs, node C (publisher) publishes that event through lightweight publish (steps 6–9). The rendezvous node performs fault tolerant actions after subscription /publish (steps 10–13).

which forwards it to the subscriber (step 7). The subscriber immediately starts a related service and notifies the broker (step 8), which then sends a notification confirmation message to the rendezvous node (step 9). The notification confirmation is a list of successfully notified subscribers and events. Data loss might occur between the publisher and broker, the broker and subscriber, or the broker and the rendezvous nodes. Using the notification confirmation message, the rendezvous node verifies that every subscriber in a cluster is successfully notified. If any subscriber is not notified, the rendezvous node communicates directly with the subscriber to immediately start its service. Furthermore, the rendezvous node communicates directly with the publisher to immediately stop its service (steps 10–13).

D. Proposed Lighteweight Rendezvous Routing Algorithm

We propose a lightweight rendezvous routing algorithm for temperature scheduling in in vivo sensors with publisher/subscriber event-based communication.

Before explaining the algorithm, we shall discuss an exist-

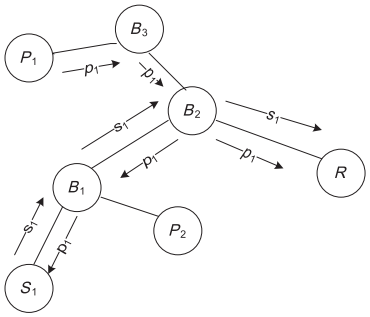

Fig. 5. Rendezvous routing algorithm is working. S, P, and B represent subscriber, publisher and broker respectively. In rendezvous routing algorithm, at first, S_1 is subscribed to B_1 and this subscription information (s_1) encounters B_2 when it is routed toward rendezvous node R. P_1 publishes event and this notification message (p_1) is routed toward R and encounters B_3, B_2. At B_2 it finds B_1's state information that matches that event. So, notification message is routed toward B_1 and at last S_1 is notified.

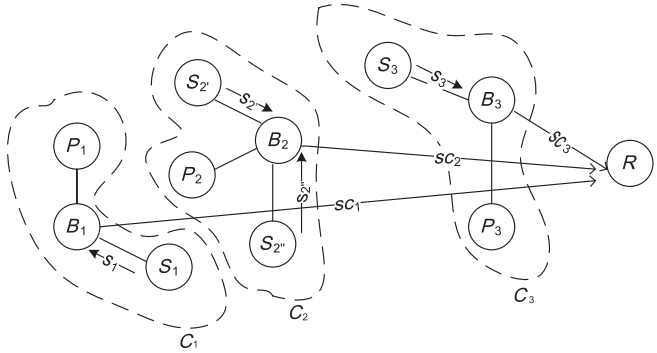

Fig. 6. Lightweight subscription is working on body sensor nodes those are divided into three clusters C_1, C_2, and C_3. In C_2, subscriber S_2' and S_2'' send subscription message s_2', s_2'', respectively, to the broker B_2. B_2 then sends subscription confirmation message (sc_2.) to rendezvous node R. Same operation goes for the clusters C_1 and C_3.

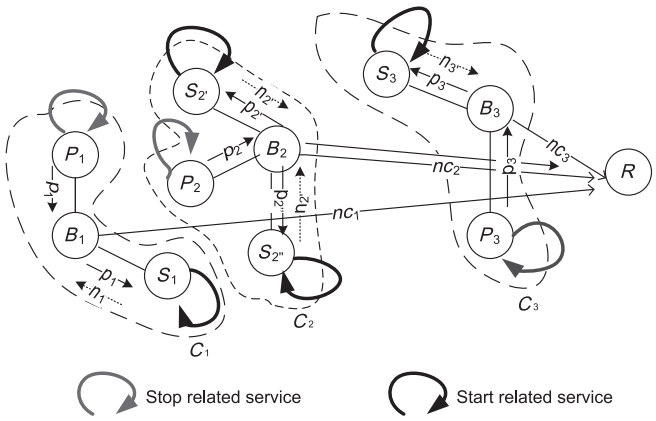

Fig. 7. Lightweight publish is working on body sensor nodes those are divided into three clusters C_1, C_2, and C_3. In C_2, when the event occurs, publisher P_2 stops related service and sends notification message (p_2) to the broker B_2. B_2 then forwards notification (p_2) to subscribers S_2' and S_2''. S_2' and S_2'' start related services immediately. Then they send confirmations (n_2' and n_2''), respectively, to the broker B_2. B_2 then sends notification confirmation (nc_2) to the rendezvous node R. Same operation goes for the clusters C_1 and C_3.

ing rendezvous routing algorithm [14]. Rendezvous-based routing algorithms, like all other content-based routing algorithms, must establish routing paths from publishers to subscribers. A publish notification message that is sent to the rendezvous node encounters all relevant subscription states in the network.

First, the subscription message is routed toward the rendezvous node and is stored at every broker along the path. Further, event notifications are routed toward the rendezvous node. When they encounter the broker corresponding to the subscribed nodes, the message is sent to the subscribe nodes.

Fig. 5 shows how rendezvous routing algorithm works. Let S, P, B, R represent subscriber, publisher, broker and rendezvous node respectively. At first, S_1 is subscribed to broker B_1. Subscription message s_1 is routed toward rendezvous node R and it encounters B_2 on its way. When P_1 publishes an event, notification message p_1 is routed toward R and it encounters B_3, B_2 on its way to R. On B_2 , it finds that state information of B_1 matches event P_1. So, notification message (p_1) is routed toward B_1 and at last subscriber S_1 is notified.

Rendezvous algorithms have some disadvantages. In particular, their operation in an implanted sensor network entails a huge overhead. In a rendezvous routing algorithm, subscription or notification messages may encounter many brokers on the path to the rendezvous node. This redundant communication is an overhead cost for low-powered implanted sensor networks.

Secondly, based on the events, any broker may function as a rendezvous node; however, other brokers must have a way to send messages to this rendezvous node. The formation of a rendezvous node depends on the routing substrate provided by a distributed hash table, which results in an overhead cost for an implanted sensor network.

Third, the fault-tolerance characteristics of rendezvous routing algorithms also result in an overhead cost. When a rendezvous node fails, another broker assumes the related responsibilities. Rendezvous nodes maintain metadata about event types, brokers, and subscribers. Metadata has to be replicated among other broker nodes so that they can function as rendezvous nodes when needed. The overhead cost of the transmission of this type of metadata is also large.

In a lightweight rendezvous routing algorithm, body sensor nodes are divided into small clusters (Figs. 6 and 7). A rendezvous node contains the cluster information and publisher, subscriber or broker role of each node in any cluster. Each cluster has a single broker node and several subscriber and publisher nodes. At first, the rendezvous node notifies the body sensor nodes about their cluster information and their roles as either subscriber, publisher or broker role (algorithm 1). Then, the lightweight subscription algorithm (Fig. 6) (algorithm 2) in each cluster creates subscriber nodes for each broker node. The broker node then sends a subscription confirmation message to the rendezvous node. The subscription confirmation message is the list of successfully subscribed nodes. The rendezvous node then verifies whether every subscriber is successfully subscribed (algorithm 3). Data loss might occur either between the subscriber and broker or between the broker and rendezvous nodes. Therefore, the rendezvous node waits for the event to occur. The rendezvous node learns of event occurrence from a no-

tification confirmation message from the lightweight publisher.

When a rendezvous node learns of an event, services on the subscribers and publishers are started and stopped, respectively. At the lightweight publisher (Fig. 7) (algorithm 4) in each cluster, when an event occurs, the publisher stops related services and sends a notification message to the broker, which then forwards it to subscribers. Subscribers immediately start the related services and notify the broker. The broker also sends notification confirmation to the rendezvous node, including the list of successfully subscribed nodes and previous events. The rendezvous node then verifies whether all subscribers in each cluster have been successfully notified (algorithm 5). Packet loss might occur between either the publisher and broker, broker and subscriber or broker and rendezvous node. In such cases, the rendezvous node directly communicates with the subscribers to start immediately their services, if they have not yet started. Moreover, the rendezvous node directly stops the services of the publishers if they have not been stopped yet.

Given a set of body sensor nodes and a rendezvous node, the body sensor nodes are divided into clusters. The nodes in each cluster can function as a publisher, subscriber or broker. A rendezvous node stores information regarding cluster formation and the roles of cluster nodes.

Let C, S, P, B, and R represent the cluster, subscriber, publisher, broker, and rendezvous nodes, respectively.

Algorithm 1 Installation at rendezvous node

1. **for all** i such that $0 \leq i \leq C$ **do**
2. **for all** j such that $0 \leq j \leq S$ **do**
3. Send cluster information and subscriber role to node j
4. **end for**
5. **for all** j such that $0 \leq j \leq P$ **do**
6. Send cluster information and publisher role to node j
7. **end for**
8. Send cluster information and broker role to a node.
9. **end for**

Rendezvous node R sends cluster information and a role (publisher, subscriber, or broker) to each node of any cluster (algorithm 1). R verifies each cluster (i) and notifies each node (j) that will function as a subscriber or publisher. R also denotes another node as the broker, which there can be one of in each cluster.

Algorithm 2 Lightweight subscription

1. **for all** i such that $0 \leq i \leq S$ **do**
2. Subscribe i to the broker with event in that cluster
3. **end for**
4. Broker sends subcription confirmation to rendezvous node R

In the lightweight subscription method (algorithm 2), each subscriber node (i) is subscribed to a broker for a specific event. Then, the broker sends a subscription confirmation to the rendezvous node R.

Algorithm 3 Post subscription at rendezvous node

1. **for all** i such that $0 \leq i \leq C$ **do**
2. **for all** j such that $0 \leq j \leq S$ **do**
3. **if** Subscription confirmation is not found for subscriber j **then**
4. (Wait for the event occurrence until the end of lightweight publish)
5. When corresponding event occurs
6. start related service of subscriber j and
7. stop related service of corresponding publisher P.
8. **end if**
9. **end for**
10. **end for**

Rendezvous node R verifies that every subscriber (j) of every cluster (i) has received a subscription confirmation (algorithm 3). If there is any subscriber missing such confirmation, R waits for the corresponding event to occur until the end of lightweight publish (algorithm 4). R learns about event occurrence from the notification confirmation found at the end of lightweight publish. When that event occurs, R communicates directly with subscriber j to start the related service and with the corresponding publisher to stop the related service.

Algorithm 4 Lightweight publish

1. **for all** i such that $0 \leq i \leq P$ **do**
2. Stop related service
3. Publish event from publisher i to broker B
4. **end for**
5. **for all** j such that $0 \leq j \leq S$ **do**
6. Forward notification from broker to subscriber j
7. **end for**
8. **for all** k such that $0 \leq k \leq S$ **do**
9. Start related service.
10. Forward confirmation from subscriber k to broker
11. **end for**
12. Send notification confirmation (the list of successfully notified subscribers and occurred events) to rendezvous node R

In the lightweight publish (algorithm 4), when an event occurs, each publisher (i) in a cluster stops the related service and sends an event notification or publish message to the broker. The broker then forwards the notification to each subscriber (j) of the cluster. Then, each subscriber (k) starts the related service and sends a confirmation to the broker. Last, the broker sends a notification confirmation message to rendezvous R, including the set of events and successfully notified subscribers.

Rendezvous node R verifies that every subscriber (j) of every cluster (i) has received a notification confirmation (algorithm 5). If there is any subscriber missing such confirmation, R communicates directly with subscriber j to start the related service and with the corresponding publisher to stop the related service.

The proposed routing algorithm is lightweight in the following ways: First, in lightweight subscription, subscribers send

Algorithm 5 Post notification at gateway node

```
for all i such that 0 ≤ i ≤ C do
    for all j such that 0 ≤ j ≤ S do
        if Notification confirmation is not found for subscriber j then
            start service of subscriber j, if not started yet and
            stop service of corresponding publisher, if not stopped yet.
        end if
    end for
end for
```

messages to a broker node in the cluster. The broker node then sends the subscribed node information to the rendezvous node. The broker does not need to communicate with other broker nodes on its way to the gateway node. The broker of each cluster separately sends subscription information to the rendezvous node. This is very lightweight compared to the redundant communication involved in the subscription of prior rendezvous algorithms. In a lightweight publisher, publishers send event notifications to the broker node in each cluster. The broker then sends notifications to corresponding subscriber nodes. Each cluster operates in the same way. A publish message does not encounter other brokers on its way to the gateway node as in prior rendezvous algorithms.

Second, the broker node of any cluster can communicate directly with the rendezvous node by a single hop. It does not need to know a path through other brokers to reach the gateway node, and so we do not need to maintain a distributed hash table for body sensor nodes with limited power.

Third, the rendezvous node is common to every cluster, and no broker node can become a rendezvous node. So, the broker node of any cluster does not need to carry the metadata of brokers of other clusters as in previous rendezvous algorithms. Thus, the overhead of metadata transmission among the brokers is not present in our proposed lightweight rendezvous routing algorithm.

Fault tolerance is performed in a lightweight manner in the proposed routing algorithm. After lightweight subscription, a broker node of a cluster sends its subscription confirmation to the rendezvous node. The rendezvous node then checks whether each subscriber has a confirmed subscription for each cluster. However, packet loss might occur between the subscriber and broker or broker and rendezvous nodes. The rendezvous node waits for the occurrence of the corresponding event. The rendezvous node learns of event occurrence from a notification message at the end of the lightweight publishing step. When the event occurs, the rendezvous node directly communicates with the subscriber and publisher to start and stop their related services, respectively. Generally, after lightweight publishing, the broker of a cluster sends notification confirmation (a list of events and successfully notified subscribers) to the rendezvous node.

The rendezvous node then checks if all subscribers in each cluster have been successfully notified. Packet loss might occur between the publisher and broker, broker and subscriber, subscriber and broker, or broker and rendezvous nodes. Therefore, if the rendezvous node receives no confirmation of the notification of any specific subscriber, it directly communicates with that subscriber to immediately start its service. At the same time, the rendezvous node communicates with the corresponding publisher node to immediately stop its service.

Table 1. Major notations used in mathematical analysis.

Parameter	Description
T	Total generated temperature
T_S	Temperature at subscription
T_P	Temperature at publish
T_{SB}	Subscriber to broker temperature
T_{BR}	Broker to rendezvous temperature
T_{PB}	Publisher to broker temperature
T_{BR}	Broker to rendezvous temperature
E	Total dissipated energy
E_S	Energy at subscription
E_P	Energy at publish
E_{SB}	Subscriber to broker energy
E_{BR}	Broker to rendezvous energy
E_{PB}	Publisher to broker energy
E_{BR}	Broker to rendezvous energy

IV. PERFORMANCE EVALUATION

A. Problem Scenario

The combination of hyperthermia, radiotherapy, and chemotherapy procedures has become the most prominent cancer treatment. The treatment depends on the temperature generated in human cells. While the temperature remains below a certain threshold, hyperthermia enhances the performance of radiotherapy and chemotherapy; however, if the temperature exceeds that threshold, the cells can be damaged, resulting in danger to the patient's health. Let us consider sensor nodes that are implanted to monitor temperature scheduling during combined hyperthermia, radiotherapy, and chemotherapy. If the temperature of a publisher node exceeds the threshold, it will direct the broker node to dissipate temperature to the subscriber node or nodes. If the temperature dissipation is not performed successfully, a remote gateway node is notified. Moreover, energy dissipation is considerable during this process; the battery power is a function of the amount of heat generated. To extend the battery life of in vivo sensors, reduction in energy consumption is important.

B. Mathematical Analysis

Total generated temperature (T) is the sum of temperature for subscription (T_S) and temperature for publish (T_P).

$$T = T_S + T_P$$

In proposed lightweight rendezvous routing algorithm, we have considered that body sensor nodes are divided into some clusters. Total subscription temperature (T_S) is the sum of subscription temperature at different clusters (T_{CS}). Total publish temperature (T_P) is the sum of publish temperature at different clusters (T_{CP}).

$$T_S = \sum_{i=1}^{c} T_{CS}$$

$$T_P = \sum_{i=1}^{c} T_{CP}$$

In each cluster, lightweight subscription temperature is represented as

$$T_{CS} = \sum_{i=1}^{s} T_{SB} + T_{BR}.$$

In each cluster, lightweight publish temperature is given by

$$T_{CP} = \sum_{i=1}^{p} T_{PB} + \sum_{i=1}^{s} T_{SB} + \sum_{i=1}^{s} T_{SB} + T_{BR}.$$

T_{SB} stands for generated temperature when packets move from a subscriber to the broker node. Temperature is generated in two ways: when packets are exhibited from subscribers and when packets are received by brokers. T_{BR} stands for temperature generated when a packet moves from a broker of each cluster to the rendezvous node. Temperature is generated when a packet is exhibited from a broker.When a packet is received by rendezvous, it's temperature increase is not considered. It is not implanted in body, it is placed outside part of body in our experiment.

T_{PB} stands for generated temperature when packets move from the publisher to the broker node. Temperature is generated in two ways: when packets are exhibited from publishers and when packets are received by brokers. T_{BS} stands for temperature generated when packets move from the broker to subscriber nodes. Temperature is also generated in two ways as in previous case.

c represents maximum number of clusters. s and p represent maximum number of subscriber and publisher nodes, respectively in a cluster.

We can rewrite the equations for energy as

$$E = E_S + E_P$$

$$E_S = \sum_{i=1}^{c} E_{CS}$$

$$E_P = \sum_{i=1}^{c} E_{CP}$$

$$E_{CS} = \sum_{i=1}^{s} E_{SB} + E_{BR}$$

$$E_{CP} = \sum_{i=1}^{p} E_{PB} + \sum_{i=1}^{s} E_{SB} + \sum_{i=1}^{s} E_{SB} + E_{BR}.$$

The main limitation of our mathematical model is that we consider a network with one gateway node and only ten body sensor nodes divided into three clusters. The number of subscribers, publishers, and brokers is each less than five. The simulations show that the same equations also hold with a similar number of publishers, subscribers, and brokers in a ten-node set. We have considered neither large clusters nor a large number of body sensor nodes. Assuming that each cluster has only one broker, if the cluster is large and the number of publishers and subscribers is high, there will be a large overhead on the broker node because all subscription and publish commands will go through a single broker node.

If there are a large number of body sensor nodes, we can consider many small clusters, each applying the lightweight rendezvous routing algorithm. Generally, only the minimum number of body sensors is used in human healthcare applications. These nodes are usually deployed in a static fashion because of physiological effects. Issues like dynamic movement and mobility of body sensor nodes are not important in such a network. Thus, decisions about cluster formation and connected coverage must be made prior to sensor network deployment. On the other hand, if a large number of body sensor nodes are necessary, it is possible to use the proposed middleware with our lightweight rendezvous algorithm. In addition, we must decide the cluster formation and connected coverage of the sensor nodes prior to installation.

B.1 Complexity Analysis

The complexity of the proposed routing algorithm can be determined in terms of the lightweight subscription and lightweight publish algorithms at the body sensor nodes and the installation, post-subscription, and post-publish mechanisms at the rendezvous node.

- Lightweight subscription: In each cluster, all subscribers send subscription message to a broker. After that, the broker sends a subscription confirmation message to rendezvous node. The complexity in each cluster depends on its total number of subscribers (N_S) and the single packet transmission from its broker to the rendezvous node. The complexity is $O(N_S) + O(1) = O(N_S)$.
- Lightweight publish: In each cluster, publishers send notification messages to a broker and then broker forwards messages to subscribed nodes. Subscribed nodes then send a confirmation message to the broker. After that, the broker sends a notification confirmation message to the rendezvous node. The complexity in each cluster depends on its total number of publishers (N_P), total number of subscribers (N_S), single packet transmission from the broker node to the rendezvous node. The complexity is $O(N_P)+O(N_S)+O(N_S)+O(1) = O(N_P) + O(N_S)$.
- Installation at rendezvous node: Rendezvous node sends cluster information and role (subscriber, publisher and broker) to body sensor nodes of each cluster. We assume that each cluster has at most one broker node. So, we assume that number of maximum clusters can be N_B. The complexity at rendezvous node depends on total number of brokers (N_B), total number of subscribers (N_S) and total number of publishers (N_P). The complexity is $O(N_B(N_S + N_P + 1)) = O(N_B(N_S + N_P))$.

Table 2. Events, subscribers, brokers, publishers used in simulation.

Parameter	Description	Value
	Number of experiments	8
	Number of total nodes in the network	10
N_B	Number of event brokers	1–5
N_S	Number of event subscribers	2–5
N_P	Number of event publishers	2–5
	Number of events	2–5
	Number of events per publisher	2–5
	Number of events per subscriber	1–5

Table 3. Simulation environment.

Parameter	Description	Value
E_P	Publisher's power	2 mW
E_S	Subscriber's power	2 mW
E_B	Broker's power	5 mW
E_{ex}	Node's exhibiting power	2 mW
E_{rec}	Node's receiving power	2 mW
E_{exR}	Node-rendezvous node(power)	5 mW
T_P	Publisher's temperature	1 U
T_S	Subscriber's temperature	U
T_B	Broker's temperature	5 U
T_{ex}	Node's exhibiting temp	1 U
T_{rec}	Node's receiving temp.	1 U
T_{exR}	Node-rendezvous node (temp.)	2 U
C_p	Threshold temperature	$46\,^{\circ}$C.
K	Thermal conductivity	0.498 [J/ms $^{\circ}$C]
T_b	Fixed blood temperature	$37\,^{\circ}$C
d	Max density	1040 kg/m^3

- Post Subscription at rendezvous node: After the broker node sends subscription confirmation message at lightweight subscription, rendezvous node checks whether for each cluster, if there are subscription confirmations for all of its subscriber nodes. If subscription confirmation is not found for any subscriber, it waits for the corresponding event-publish. When the event is published, rendezvous node immediately starts and stops related services of subscriber and publisher respectively. The complexity at gateway node depends on total number of brokers (N_B) and total number of subscribers (N_S). The complexity is $O(N_B N_S)$.
- Post Publish at rendezvous node: After the broker node sends notification confirmation message at lightweight publish, rendezvous node checks whether for each cluster, if there are notification confirmations for all of its subscribed nodes. If notification confirmation is not found for any subscriber, rendezvous node immediately starts and stops related services of subscriber and publisher respectively. The complexity at gateway node depends on total number of brokers (N_B) and total number of subscribers (N_S). The complexity is $O(N_B N_S)$.

C. Simulation Environments

We have performed our extensive simulation in a JAVA program. Tables 2 and 3 show simulation environment used. Event type is unique and it occurs when the temperature of node exceeds threshold value of $46\,^{\circ}$C.

Our simulations consist of several cases. In observation 1, we compare our proposed lightweight rendezvous algorithm (LR) with a rendezvous routing protocol in a 6×6 mesh topology. In observation 2, we compare the performance of LR with different randomly distributed node orientations. In observation 3, we compare the proposed LR with directed diffusion, flooding, and omniscient multicasts in a 12×12 mesh topology simulation. Each experimental set consists of ten in vivo sensor nodes and a rendezvous node. The simulation steps are as follows.

C.1 Subscribe

In each cluster, subscriber nodes subscribe to a broker. The broker then confirms subscriptions with the rendezvous node.

C.2 Publish Least Fault Tolerant

We considered the least fault-tolerant cases of both the lightweight rendezvous and rendezvous routing algorithms.

In the lightweight publisher, when a broker does not receive confirmation from a subscribed node, the broker does not send notification to the rendezvous node. Packet transfers from subscribers to the broker or from the broker to the rendezvous node do not occur. However, fault tolerance is compromised because the rendezvous node cannot perform actions according to the post-notification algorithm.

In the rendezvous routing algorithm, when a broker does not receive cluster information from other broker nodes, considerably less heat is generated. However, fault tolerance is compromised because when a broker does not have a path to another broker, it may miss its path to the rendezvous node.

C.3 Publish Most Fault Tolerant

We also considered the most fault-tolerant cases for both the lightweight rendezvous and rendezvous routing algorithms.

In the lightweight rendezvous routing algorithm, a broker receives confirmation from the other subscribed nodes at the lightweight publish step. The broker sends a notification message to the rendezvous node. The rendezvous node can then take action according to the post-notification algorithm if there is any packet loss between either the publisher and subscriber, subscriber and broker, or broker and rendezvous nodes. Fault tolerance is present, but it involves more heat generation in the broker because of the packet transmission from the subscriber to the broker and from the broker to the rendezvous node.

In rendezvous routing, every broker sends cluster information to other brokers on its way to the rendezvous node. This generates large quantities of heat owing to the large packet sizes; however, it also ensures that a broker can find a path to the rendezvous node in case of packet failure.

D. Performance Metrics

Three performance metrics were considered: Generated heat, dissipated energy, and delay.

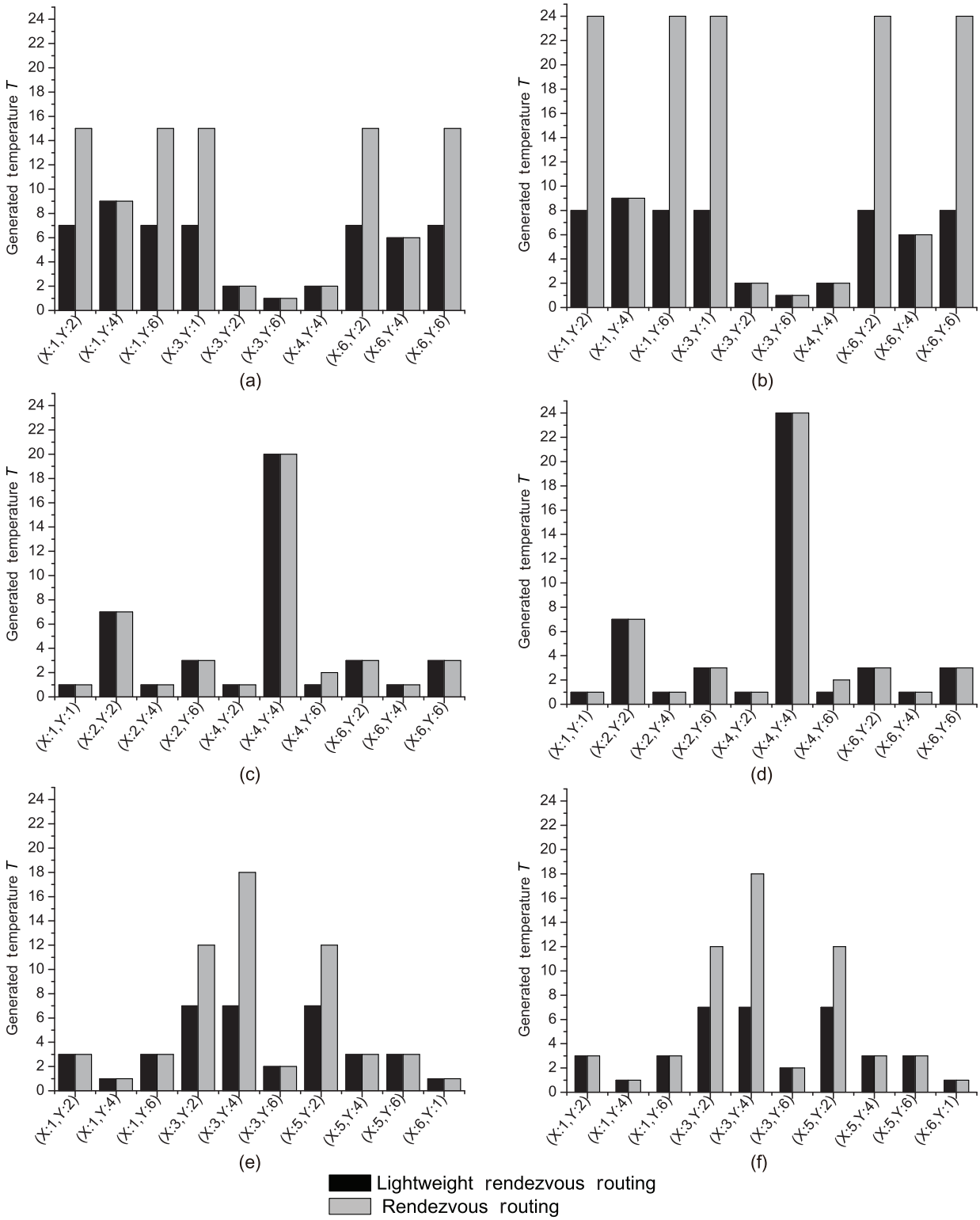

Fig. 8. Comparison on generated temperature on 3 different sets of 10 in vivo sensor nodes using LR and rendezvous routing algorithm(RR) in both the least-fault-tolerant (left side) and the most-fault-tolerant(right side) situtations. In each figure, LR generates less temperature than RR in broker nodes, where other nodes generate equal temperature. Node orientation (3 publishers, 4 subscribers, and 3 brokers) representing Figs. 8(e) and 8(f) is the most desirable because it generates the least temperature using LR. This also generates almost same temperature in both the least and the most fault tolerant states using LR. Node orientation for Fig. 8(c) and 8(d) is not acceptable because it generates huge temperature in the single broker node and it can be dangerous for surrounding human tissues. Most-fault-tolerancy (right side) causes much more temperature in brokers of RR than brokers of LR.

D.1 Generated Heat

Temperature scheduling is a major concern in the treatment of cancer. The in vivo sensor nodes generate heat, and if we deploy the nodes in our proposed lightweight routing algorithm for event-based communication, more heat will be generated. Thus, we analyzed the overall increase in temperature in the in vivo sensor nodes using our routing algorithm. It is worth mentioning that because the rendezvous node is positioned outside the body, the temperature at the rendezvous node was not considered.

D.2 Dissipated Energy

In vivo sensor nodes can store limited energy and have a short battery life. The more the heat generated, the greater is the battery usage (including additional recharging), and hence, we must ensure that our algorithm dissipates as little energy as possible.

D.3 Delay

Delay is the one-way latency observed from the time of the subscription of all events to the time of successful publication of those events. We compared our proposed protocol with directed diffusion, flooding, and omniscient multicast protocols. With these methods, the delay is the total time required from the transmission of events from source nodes to their arrival at sink nodes

E. Simulation Results

E.1 Observation 1

Fig. 8 shows the temperatures generated for ten in vivo sensor nodes using the proposed lightweight rendezvous routing and conventional rendezvous routing algorithms. These nodes are deployed in a 6×6 mesh topology. The black and red bars represent the proposed algorithm and the conventional rendezvous routing algorithm, respectively. In each row, the figures in the left and the right panels represent the least and the most fault-tolerant cases, respectively. We refer to the proposed lightweight rendezvous algorithm and the conventional rendezvous routing algorithm as LR and RR, respectively.

Figs. 8(a) and 8(b) show the temperatures with three nodes coordinates (X:3,Y:2), (X:3,Y:6), (X:4,Y:4) in a publisher role, two nodes coordinates (X:1,Y:4), (X:6,Y:4) in a subscriber role, and five nodes in a broker role coordinates (X:1,Y:2), (X:1,Y:6), (X:3,Y:1), (X:6,Y:2), (X:6,Y:6). In both the figures, the five brokers generate less heat in LR than in RR because in LR, the brokers do not need to send subscription or publish messages to other brokers since they communicate directly with the rendezvous node. However, in RR, brokers need to send cluster information to every other broker on their way to the common rendezvous node.

Figs. 8(c) and 8(d) show the temperatures with five nodes (X:1,Y:1), (X:2,Y:4), (X:4,Y:2), (X:4,Y:6), (X:6,Y:4) in a publisher role, four nodes (X:2,Y:2), (X:2,Y:6), (X:6,Y:2), (X:6,Y:6) in a subscriber role, and one node in a broker role (X:4,Y:4). The two protocols produce the same temperature because in RR, a single broker does not need to send messages to other brokers on its way to the rendezvous node. In both the figures, the temperature at the broker node (X:4,Y:4) is very high as compared to that at other nodes, which may harm the surrounding human tissues.

Table 4. Permutation of publisher, subscriber and broker nodes in simulation.

P	S	B
3	2	5
2	3	5
3	3	4
3	4	3
4	3	3
4	4	2
5	4	1
4	5	1

Figs. 8(e) and 8(f) show the temperature with three nodes (X:1,Y:4), (X:3,Y:6), (X:6,Y:1) in a publisher role, four nodes (X:1,Y:2), (X:1,Y:6), (X:5,Y:4), (X:5,Y:6) in a subscriber role, and three nodes in a broker role (X:3,Y:2), (X:3,Y:4),(X:5,Y:2). In both the figures, LR generates considerably less heat than RR because with LR, the three broker nodes communicate directly with the rendezvous node, while in RR, the broker nodes communicate with other broker nodes on their way to the rendezvous node.

In all the figures, the least fault-tolerant case (left panel) is similar to the most fault-tolerant case (right panel), with the exception of the broker nodes. For LR, the broker nodes in the most fault-tolerant setup generate slightly more heat than those in the least fault-tolerant because in LR, each broker node receives a single confirmation message from each subscriber. As a result, only the broker's temperature is increased by a small amount. With RR, the broker nodes in the most fault-tolerant setup generate significantly more heat because each broker receives a confirmation message from all other brokers. Generally, in RR, the confirmation message from each broker contains cluster state information, and the packet size is large; this causes the brokers to generate substantially more heat.

In all scenarios, three publishers, four subscribers, and three brokers (Figs. 8(e) and 8(f)) generate the lowest temperature using LR. Both the least fault-tolerant and most fault-tolerant states generate almost the same temperature in the nodes, and the temperature at any broker node is not much higher than that at the other subscriber or publisher node. Thus, the orientation with three publishers, four subscribers, three brokers, and the LR algorithm significantly decreases the temperature in implanted sensor nodes.

E.2 Observation 2

Fig. 9 shows the total temperature and energy dissipation for all in vivo sensor nodes in all node orientations of Table 4 when using LR. We considered both the least and most fault-tolerant cases.

Fig. 9(a) shows the total heat generated for different node orientations. The maximum temperature is generated when half of the nodes are brokers and there are an equal number of

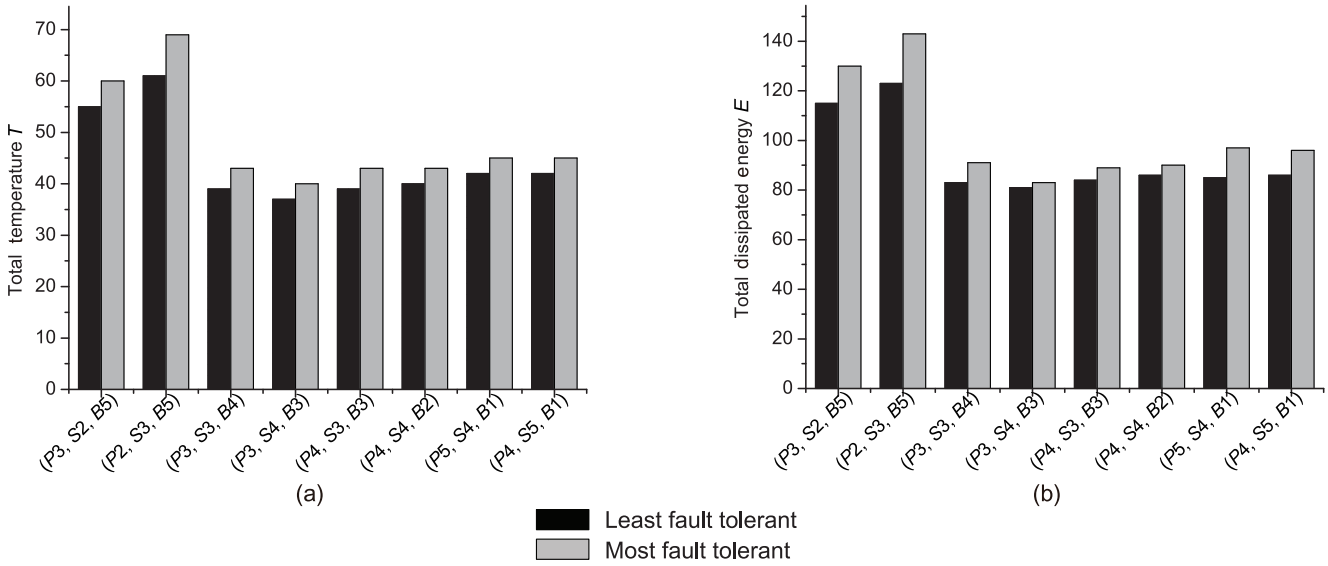

Fig. 9. (a) Total temperature and (b) total energy dissipation in different node orientations using lightweight rendezvous routing algorithm.

subscribers and publishers. Reducing the total number of brokers also reduces the final temperature. The temperature is minimized when the node distribution is approximately balanced (e.g., three publishers, four subscribers, and three brokers). However, if we reduce the number of brokers such that we have only one broker and all other nodes are subscribers or publishers, the temperature is decreased. The main concern is that the temperature of the single broker is very high. This can be very dangerous for nearby cells that may be damaged due to thermal effects.

Fig. 9(b) shows the total energy dissipated by our proposed routing algorithm. With three publishers, four subscribers, and three brokers, the least and most fault-tolerant cases dissipate similar amounts of energy, less than that with other node orientations.

To summarize, our simulations show that an orientation with three publishers, four subscribers, and three brokers provides the best results with our lightweight rendezvous routing algorithm.

E.3 Observation 3

Next, we compared our proposed lightweight rendezvous routing algorithm with flooding, omniscient multicast, and directed diffusion protocols [20] by implementing all four of them as a Java program. We chose two performance metrics, delay and dissipated energy. We used ten nodes with four subscribers, three publishers, and three brokers in a 12 cm$\times$12 cm square. The subscribers were placed at the bottom and top of the square, with the publishers and brokers in the middle. A subscriber/publisher can be considered as source/sink or sink/source at the time of subscription or publication, respectively. The broker can be considered as the interim node used in directed diffusion, multicast, or flooding. Each node has a transmission range of 2 cm. To calculate the delay and dissipated energy, we executed one of the four protocols among the nodes for 120 s, with four events generated every 30 s. Packet transmission among the in vivo sensor nodes required one unit of energy, while packet transmission between the in vivo sensor node and the rendezvous node required two units of energy.

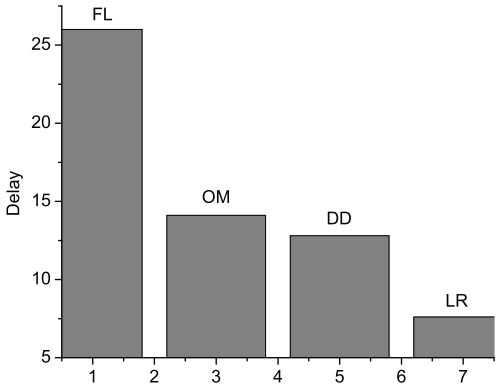

Fig. 10. Delay comparision of flooding (FL), omniscient multicast (OM), directed diffusion (DD), and lightweight rendezvous routing (LR) algorithm.

Fig. 10 compares the delays under the various schemes. In the flooding scheme, the delay is very high since it takes time for every source to send packets to every other node. For the omniscient multicast scheme, the delay is relatively small since the sources use the shortest paths to send packets to the sinks. Direct diffusion shows slightly less delay with the advantages of aggregation and in-node processing. In the proposed lightweight rendezvous routing algorithm, the delay is very small compared to the others because in each cluster, nodes require relatively few hops to send a packet to the rendezvous node.

Fig. 11 compares the energy dissipated by all four methods. Fig. 11(a) shows that the flooding scheme with sources placed at the bottom and top of the square floods all events to every node in the network. The nodes placed in the middle of the square dissipate more energy because there is relatively higher packet transmission in that region. Fig. 11(b) shows that with the omniscient multicast scheme, each source generates the shortest path multicast tree to all sinks. All multicast trees use a common interim node to create the shortest path, and hence, energy dissi-

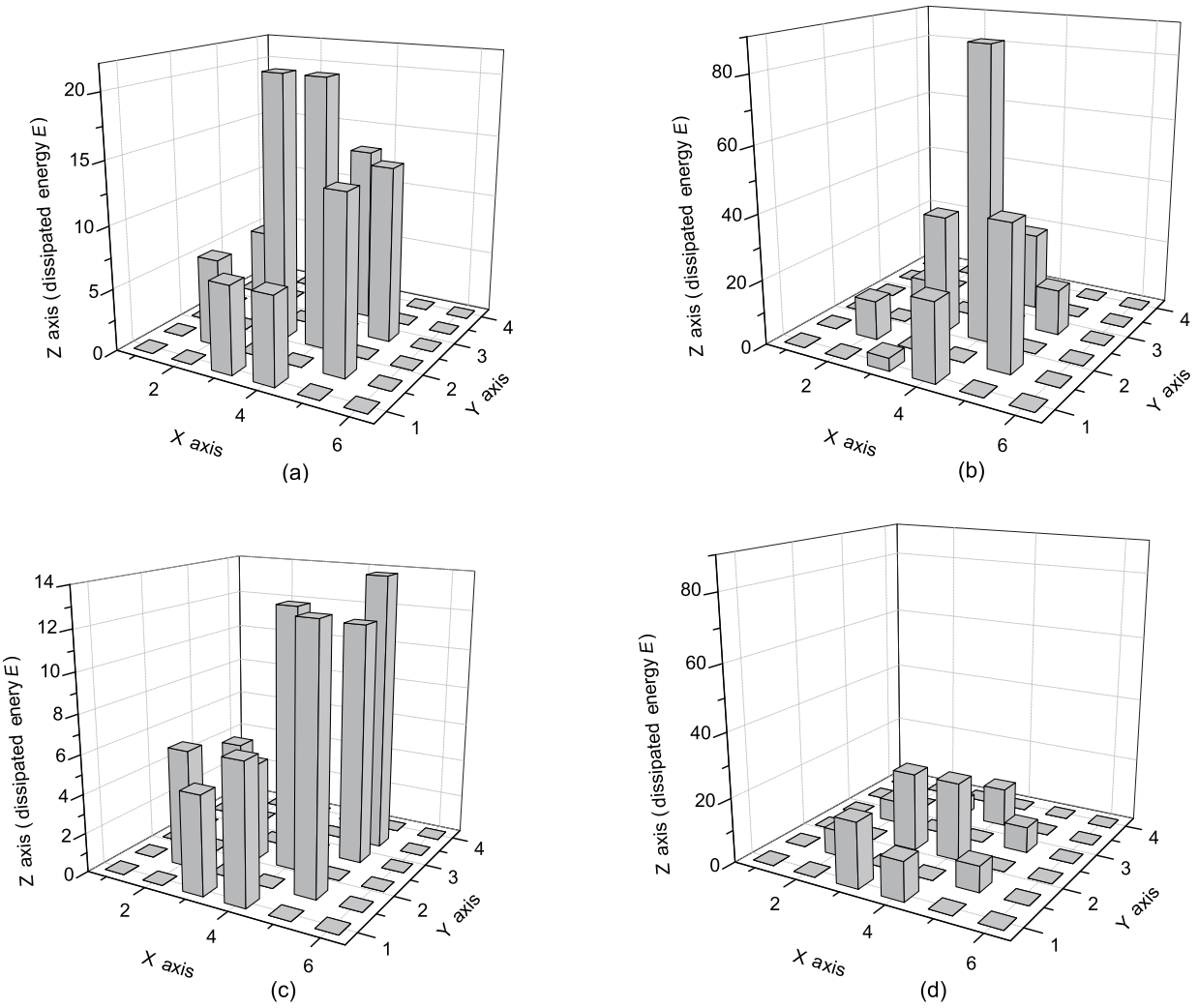

Fig. 11. Energy dissipation in a set of 10 body sensor nodes using (a) flooding, (b) omniscient multicast, (c) directed diffusion, and (d) lightweight rendezvous routing algorithm.

pation for that node is large. Fig. 11(c) shows that with directed diffusion, energy dissipation is low as compared to that with flooding and multicast schemes. Interests are propagated from sources to sinks, placed at the middle positions using interim broker nodes, and events are propagated according to the reverse path. Fig. 11(d) shows that the lightweight rendezvous routing algorithm dissipates relatively less energy as compared to the previous three protocols. The nodes are divided into three clusters, with the lightweight publish-subscribe mechanism working separately in each cluster. As brokers are involved in subscription, publication, and communication with the rendezvous node, energy dissipation is higher in these broker nodes.

V. RELATED WORKS

A. Temperature-Aware Routing Algorithms

Temperature-aware routing algorithms [4], [5] that generate less heat have been proposed in the past, but they all suffer from complexity overhead. Existing middleware for BSNs [10], [11] uses gateway approaches in which body sensor nodes are in contact with a gateway device. Grid middleware approaches for in vivo sensor nodes have not been extensively studied, although [17] discusses the prospects of sensor networks in cancer hyperthermia treatment without making any conclusive proposal.

B. OSGI and its Implementation in Distributed and Sensor Environments

OSGI [15] is an open specification for the delivery of multiple services over wide area networks to local networks and devices. It is an open framework that enables multiple software services to be loaded and run on a service gateway. The OSGI specification supports easy maintenance of devices and services, dynamic update of device drivers, and easy of delivery of services.

Research is ongoing to take advantage of both central module management and distributed deployment. R-OSGI [18] is a distributed middleware platform that extends the centralized OSGI platform to support distributed module management. For developers, OSGI is a conventional module management system; however, at runtime, it serves as a distributed application with its installed modules. However, R-OSGI uses the Java runtime environment as in the core OSGI specification.

Efforts have been made to deploy OSGI in sensor net-

works. [19] presents an OSGI middleware component called CITIC that attempts to provide scalability, easy integration, and dynamic deployment of wireless sensor devices into ambient assisted living environments [20], [21]. [18] aimed to bridge the gap between high-end network devices and resource-constrained sensors with a service-oriented architecture built on OSGI. All of these use JVM at runtime. However, component-based nesC over TinyOS is still the most feasible OS for sensor networks, and JVM solutions are not efficient for resource-constrained implanted sensor nodes. In our previous work [12], we have shown how OSGI can be used in a grid middleware for BSNs. Our proposed middleware uses OSGI with a C runtime environment deployed in implanted sensor nodes.

VI. CONCLUSION

Overlay broker network routing incurs an overhead cost for in vivo sensor nodes. In this paper, we use a new approach consisting of a grid middleware component with a lightweight rendezvous algorithm as an alternative. Our routing algorithm schedules temperatures in in vivo sensor nodes to facilitate combined hyperthermia, radiotherapy, and chemotherapy cancer treatment. We have considered experimental sets consisting of ten nodes. In future work, we propose to study additional issues related to lightweight broker network routing, including scalability and node orientation.

REFERENCES

[1] X. Fu, W. Chen, S. Ye, Y. Tu, Y. Tang, D. Li, H. Chen, and K. Jiang, "A wireless implantable sensor network system for in-vivo monitoring of physiological signals," *IEEE Trans. Inf. Technol. Biomed.*, vol. 15, no. 4, pp. 577–584, July 2011.

[2] R. Kamal, M. O. Rahman, and C. S. Hong, "A lightweight temperature scheduling routing algorithm for an implanted sensor network," in *Proc. ICTC*, Korea, Sept. 2011, pp. 396–400.

[3] R. A. Weienberg, *The Biology of Cancer.* Garland Science, 2006.

[4] G. Lazzi, "Thermal effects of bioimplants," *IEEE Eng. Med. Biol. Mag.*, 2005.

[5] N. Das, P. Ghosh, and A. Seni, "Approximation algorithm for avoiding hotspot formation of sensor networks for temperature sensitive environments," in *Proc. IEEE GLOBECOM*, vol. 5, USA, 2009, pp. 4566–4571.

[6] P. Bellavista and A. Corradi, *The Handbook of Mobile Middleware.* Auerbach Publications, 2007.

[7] S. Tarkoma, *Mobile Middleare Architecture, Patterns and Practice.* John Wiley and Sons Ltd, 2009.

[8] G. Z. Yang and M. Yacoub, *Body Sensor Networks.* Springer Verlag, 2006.

[9] M. A. Razzaque, C. S. Hong, and S. Lee, "Data-centric multiobjective QoS-aware routing protocol for body sensor networks," *Sensors*, vol. 11, no. 1, pp. 917–937, Jan. 2011

[10] S. L. Keoh, N. Dulay, and S. Filho, "Self-managed cell: A middleware for managing body sensor networks," in *Proc. MobiQuitous*, USA, Aug. 2007.

[11] A. B. Waluyo, I. Pek, X. Chen, and W. S. Yeoh,"Design and evaluation of lightweight middleware for personal wireless body area network," *Personal and Ubiquitous Computing*, Springer-Verlag, 2009.

[12] R. Kamal, M. A. Razzaque, and C. S. Hong, "Grid middleware for invivo sensor nodes," in *Proc. ICOIN*, pp. 200–205, Malaysia, Jan. 2011, pp. 200–205.

[13] G. Muhl, L. Fiege, and P. R. Pietzuch, *Distributed Event-Based Systems.* Springer-Verlag Berlin Heidelberg, 2006.

[14] P. R. Pietzuch, "Hermes: A scalable event-based middleware," *30th Technical Report, Number 590, University of Cambridge*, ISSN 1476–2986, 2004.

[15] OSGI Alliance, OSGI-open service gateway initiative. [Online]. Available: http://www.osgi.org

[16] M. O. Rahman, M. M. Alam, M. M. Monowar, C. S. Hong, and S. Lee, "nW-MAC: Multiple wake-up provisioning in asynchronously scheduled duty cycle MAC protocol for wireless sensor networks," *Ann. Telecommun.*, vol. 66, no. 9–10, pp. 567–582, Oct. 2011.

[17] V. R. Singh and K. Singh, "Wireless sensor networks for biomedical applications in cancer hyperthermia," in *Proc. 30th IEEE EMBC*, Canada, Aug. 2008.

[18] J. S. Rellermeyer, G. Alonso, and T. Roscoe, "R-OSGi: Distributed applications through software modularization," in *Proc. ACM/IFIP/USENIX Middleware*, USA, Oct. 2007.

[19] J. Leguay, M. LopezRamos, K. JeanMarie, and V. Conan, "An efficient service oriented architecture for heterogeneous and dynamic wireless sensor networks," in *Proc. IEEE LCN*, Canada, Oct. 2008, pp. 740–747.

[20] C. Y. Poon, Q. Liu, H. Gao, W.-H. Lin, and Y.-T. Zhang, "Wearable intelligent systems for e-health," *J. Comput. Sci. Eng.*, vol. 5, no. 3, pp. 246–256, Sept. 2011.

[21] S. Helal, R. Bose, C. Chen, A. Smith, S. D. Deugd, and D. J. Cook, "STEP-STONE: An intelligent integration architecture for personal tele-health," *Journal of Computing Science and Engineering*, vol. 5, no. 3, pp. 269–281, Sept. 2011.

[22] C. Intanagonwiwat, D. Estrin, J. Heidemann, and F. Silva, "Directed diffusion for wireless sensor networking," *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 2–16, Feb. 2003.

Rossi Kamal received his B.Eng. degree in Computer Science and Engineering Department of University of Dhaka, Bangladesh, in 2009. He is working for his Ph.D. degree in Department of Computer Engineering at Kyung Hee University, Korea since March, 2010. His research area includes design and software-application of optimization tools, machine learning, evolutionary algorithms for communication specially healthcare, M2M networks. He has also been involved in a software development project for M2M networks and services since 2010.

Nguyen H. Tran received the B.S. degree in Electrical Engineering at Hochiminh City University of Technology in 2005. He was then awarded the Korean IITA Fellowship for his graduate study at Kyung Hee University, where he received the combined M.S. and Ph.D. degrees in Computer Engineering in 2011 with Best Foreign Student Award. During 2011, he was a Postdoctoral Research Associate in Kyung Hee University. He joined the Department of Computer Engineering, Kyung Hee University in 2012, where he is currently an Assistant Professor. His research interest is mainly on employing various applied mathematical tools such as queueing theory, optimization theory, control theory and game theory to design, analyze, and optimize the cutting-edge applications in stochastic communication networks.

Choong Seon Hong received his B.S. and M.S. degrees in Electronic Engineering from Kyung Hee University, Seoul, Korea, in 1983, 1985, respectively. In 1988 he joined KT, where he worked on Broadband Networks as a member of the technical staff. From sept. 1993, he joined Keio University, Japan. He received the Ph.D. degree at Keio University in March 1997. He had worked for the Telecommunications Network Lab. KT as a Senior Member of technical staff and as a director of the networking research team until August 1999. Since September 1999, he has worked as a professor of the Department of Computer Engineering, Kyung Hee University. He has served as a Program Committee Member and an Organizing Committee Member for International conferences such as NOMS, IM, APNOMS, E2EMON, CCNC, ADSN, ICPP, DIM, WISA, BcN, TINA, SAINT, and ICOIN. His research interests include ad hoc networks, power line communications, network management, and future internet. He is a Senior Member of IEEE and Member of ACM, IEICE. IPSJ, KIISE, KICS, and KIPS.