Future Generation Computer Systems 78 (2018) 626–640

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A reliable IoT system for Personal Healthcare Devices

Min Woo Woo, JongWhi Lee, KeeHyun Park *

Computer Engineering Department, Keimyung University, Republic of Korea

ARTICLE INFO

Article history:
Received 3 June 2016
Received in revised form
7 March 2017
Accepted 1 April 2017
Available online 12 April 2017

Keywords:
IoT
Personal Healthcare Device
OneM2M
u-healthcare
Fault-tolerant
Daisy chain

ABSTRACT

Healthcare applications in IoT systems have been receiving increasing attention because they help facilitate remote monitoring of patients. In this paper, we propose a reliable oneM2M-based IoT system for Personal Healthcare Devices. In order to use a Personal Healthcare Device as an Application Dedicated Node in the proposed system, a protocol conversion between ISO/IEEE 11073 protocol messages and oneM2M protocol messages is performed in gateways located between Personal Healthcare Devices and the PHD management server. The proposed oneM2M-based IoT system for Personal Healthcare Device is constructed, and evaluated in various experiments. The experiments show that the protocol conversion performs effectively, and that the conversion process does not cause the system to suffer serious performance degradation, even when the number of Application Dedicated Node is quite large.

Some Personal Healthcare Device data is too precious to lose due to system failures under u-healthcare environments. However, until now, few studies have focused on fault-tolerant health data services. Therefore, we also propose a fault-tolerant algorithm for the reliable IoT system in which gateways on the same layer in the system are linked to form a daisy chain for fault tolerance at the level, and a gateway stores the backup copy of the previous gateway positioned immediately ahead of the gateway in the daisy chain. The upper-layered gateway stores the parity data of the daisy chain as well. In this manner, as many as two gateway faults occurred at the same time can be recovered. For experiments, the resource trees of the oneM2M-based IoT system were expanded to store information on daisy chains, backup copies, and parity. Our experiments reveal that the proposed algorithm can recover from faults on gateways in the oneM2M-based IoT system.

1. Introduction

Healthcare applications in IoT (Internet of Things) [1–6] systems have begun to draw attention recently, because IoT systems provide many useful features that facilitate remote monitoring of patients [7,8]. A Personal Healthcare Device (PHD) becomes an essential part of a remote monitoring system when healthcare applications in IoT systems are considered. PHDs are portable electronic healthcare devices that sense and measure users' biomedical signals. As people monitor their health more carefully than ever, PHDs will become increasingly popular, and must be able to seamlessly connect to main healthcare servers [9–12].

System failures [13–17] may occur because of hardware malfunctions, software bugs, power shortages, or environmental hazards. Most studies on IoT systems have been conducted assuming that few faults exist through which the operations of an IoT system are disrupted. However, because sensors or devices of IoT systems are geographically distributed and rarely maintained, these systems are increasingly vulnerable to failures such as power shortages or environmental hazards than are other systems. Moreover, as the number of nodes in a large-scale IoT system increases, the possibility of fault occurrence increases, causing the system to work improperly. Moreover, some PHD data is too precious to lose due to system failures under u-healthcare environments. However, until now, few studies have focused on fault-tolerant health data services in IoT environments.

The purpose of this study is twofold. The first purpose is to propose and construct an IoT system for PHDs based on the oneM2M communication protocol [5,6]. Whereas (programs installed on) sensors or meters can be Application Entities (AEs) in most oneM2M systems, (programs installed on) PHDs are the Application Entities (AEs) in the oneM2M system proposed in this paper. In order to use PHDs in a oneM2M system, a communication protocol conversion process is needed because PHDs and IoT systems use different communication protocols. In other words, the ISO/IEEE 11073 protocol [9,10] is an international standard

* Correspondence to: Computer Engineering Department, Keimyung University, 1000 Sindang-dong, Dalseo-gu, Daegu 704-701, Republic of Korea.
E-mail addresses: wmwpgm@gmail.com (M.W. Woo), dragon8829@naver.com (J. Lee), khp@kmu.ac.kr (K. Park).

http://dx.doi.org/10.1016/j.future.2017.04.004
0167-739X/

for PHD communication, while the oneM2M protocol is an international standard for the IoT system considered in this paper.

There may be hundreds of IoT servers running in this world, and the number of such IoT servers will increase greatly every year. It would hardly be possible to use IoT systems if every IoT server supported its own proprietary communications protocol only. Therefore, standard communications protocols have been proposed in IoT environments for interoperability, and most the IoT servers have been made to support the standard communications protocols. On the other hand, most PHDs do support the ISO/IEEE 11073 communications protocol because the protocol is a standard communications protocol for PHDs. This is the reason why a protocol conversion process is needed in the paper. Until now, few studies have focused on protocol conversion process on health data services in IoT environments. In this study, on the basis of our previous protocol conversion study [18], the protocol conversion process is restructured and reprogrammed to allow its application to a wider variety of PHDs. Some experiments are performed on the prototype of the proposed system, to ensure that the system does not suffer serious performance degradation when the number of PHDs is quite large.

The second purpose is to propose a fault-tolerant algorithm for the reliable IoT system. We propose a fault-tolerant algorithm in which gateways on the same layer in the system are linked to form a daisy chain for fault tolerance at the level, and a gateway stores the backup copy of the previous gateway positioned immediately ahead of the gateway in the daisy chain. The backup copy of the last gateway in the daisy chain is stored by the upper-layered gateway in the system. The upper-layered gateway stores the parity data of the daisy chain as well. In this manner, as many as two gateway faults occurred at the same time can be recovered. The fault-tolerant algorithm that employs the daisy chain proposed in this study is evaluated based on experiments on the multilayered oneM2M-based IoT system. For experiments, the resource trees of gateways and the server are expanded to store fault-tolerant-related data such as daisy chain data, backup copies, and parity data. Our experiments reveal that the proposed algorithm can recover from faults on gateways in the oneM2M-based IoT system.

Healthcare applications in IoT systems have begun to draw attention recently because they provide many features that are useful for remote monitoring of patients, including scalability, flexibility, and interoperability [7,8,19]. When healthcare applications in IoT systems are considered, gateways located between sensors (or PHDs) and the IoT servers usually play very important roles [8]. Good (Poor) management of the gateways usually leads to good (poor) performance in the entire IoT system. Thus, most of the protocol conversion process and fault-tolerance process proposed in this study are performed at the gateways.

The remainder of this paper is organized as follows. Section 2 describes some related studies, and Section 3 explains the structure of the oneM2M-based IoT system and modules constructed in this study Section 4 discusses communication protocol conversion mechanisms between oneM2M protocol messages and ISO/IEEE 11073 protocol messages. Section 5 shows the results of some experiments using the system constructed in this study, along with a discussion based on the results. Section 6 discusses the proposed fault-tolerant algorithm for the reliable IoT system Finally, Section 7 draws some conclusions and discusses some possible directions for future research.

2. Related studies

The ISO/IEEE 11073 communication protocol [9,10] was proposed by an ISO/IEEE committee as an international standard to provide interoperability for health and medical services in ubiquitous environments (especially using PHDs). The oneM2M communication protocol is an international standard for IoT systems [5,6]. In such a system, a sensor or device (i.e., an installed program on either) represents an application dedicated node-application entity (ADN-AE) that gathers surrounding data and transmits them to the system's middle node-common service entity (MN-CSE). An MN-CSE controls or monitors ADN-AEs that belong to the MN-CSE. Moreover, it performs processing that is necessary to achieve efficient communication between ADN-AEs and the infrastructure node-common service entity (IN-CSE). A manager or user can access data stored in the IN-CSE through an ADN-AE.

In [11], a message processing scheme for an integrated PHD gateway in an integrated PHD management system is proposed. The ISO/IEEE 11073 communication protocol is used to transmit health messages measured by a PHD to the integrated PHD management server via the related integrated PHD gateway. The OMA DM communication protocol is used to transmit device management commands issued by the integrated PHD management server to a PHD via the related integrated PHD gateway. In [10], a multilayer secure biomedical data management system for managing a very large number of diverse personal health devices is proposed. The ISO/IEEE 11073 protocol and OMA DM protocol are extended and implemented in the system. The PHD gateway module receives separate ISO/IEEE 11073 or OMA DM messages from the PHD agents of the PHDs in order to integrate them to send the server a single integrated message.

An IoT application that has emerged is healthcare [7,8,19,20]. The importance of gateways located between sensors and the Internet has been recognized in IoT-based patient monitoring systems, because the gateways have beneficial knowledge and constructive control over both the sensor network and the data to be transmitted over the Internet [8]. The Smart e-Health Gateway proposed in [8] provides local storage to perform real-time local data processing and embedded data mining.

When a patient's biomedical data is processed, reliable IoT systems should be provided to facilitate fault-tolerant healthcare services. Until now, few studies have focused on fault-tolerant health data services. Studies on fault-tolerant IoT systems have mainly focused on routing problems [20–22]. In [20], a fault-tolerant and scalable IoT architecture for healthcare is proposed. Fault tolerance is achieved via backup routing between nodes and advanced service mechanisms, to maintain connectivity in the presence of faults on the paths between system nodes. A fault-tolerant routing protocol for IoT systems is proposed to assure successful delivery of packets, even in the presence of faults on the paths between a pair of source and destination nodes [21]. The proposed approach based on the learning automata and cross-layer concepts dynamically selects the optimum path. Dijkstra's algorithm can be used to select secure fault tolerant routing paths to enhance performance and minimize energy consumption [22].

3. Structure of a oneM2M-based IoT system for PHDs

3.1. System structure

Fig. 1 shows the structure of the proposed oneM2M-based IoT system for PHDs. In an IoT system, (a program installed on) a sensor or device represents an Application Dedicated Node-Application Entity (ADN-AE) that gathers surrounding data and transmits it to the system's Middle Node-Common Service Entity (MN-CSE). A (program installed on a) PHD acts as an ADN-AE in the proposed system. An MN-CSE controls or monitors ADN-AEs that belong to the MN-CSE; moreover, it performs processing that is necessary to achieve efficient communication between ADN-AEs and the Infrastructure Node-Common Service Entity (IN-CSE).

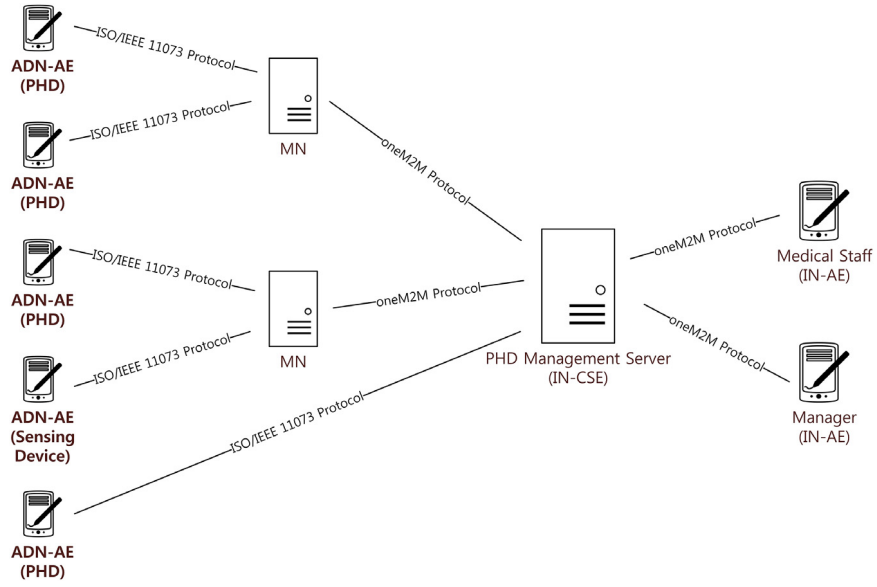

Fig. 1. Structure of the oneM2M IoT system for PHDs.

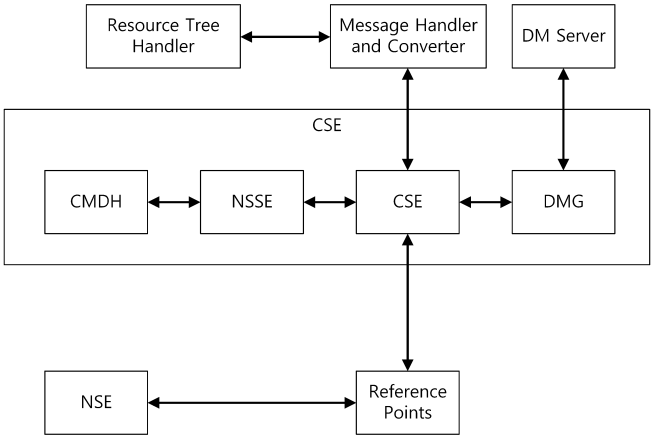

Fig. 2. CSE structure.

Because the ISO/IEEE 11073 communication protocol is an international standard for PHD communication, the protocol is used between PHDs (ADN-AE) and gateways (MN-CSE) in the proposed system; the oneM2M protocol is used between PHD management servers (IN-CSE) and MN-CSEs in ordinary IoT systems. The IN-CSE controls and monitors all the ADN-AEs and MN-AEs directly or indirectly. In addition, all the processed data is stored in the IN-CSE. In the proposed system, the PHD management server performs the IN-CSE's role. In the proposed system, managers, medical staff, and PHD users can access biomedical data from the PHD management server. Because the oneM2M protocol is used for communication between the IN-CSE and an MN-CSE, one of the MN-CSE's responsibilities is to convert the ISO/IEEE 11073 protocol into the oneM2M protocol and vice versa.

3.2. CSE structure

Fig. 2 shows the CSE structure for the MN-CSEs and IN-CSEs constructed in this study. The system is implemented in C#.

Some of the important system modules constructed in this study are as follows:

- CMDH (Communication Management/Delivery Handling) provides data delivery service. It determines when and how data is delivered.
- DMG (Device Management) provides device management service.
- NSSE (Network Service Exposure/Service Execution and Triggering) controls communication related to the base network. It also provides network access via an MCN reference point.
- Resource Handler manages the resource tree that stores information for all objects managed by the system.
- Message Handler and Protocol Converter analyzes received messages, in order to execute operations in the message. It also converts ISO/IEEE 11073 messages into oneM2M messages and vice versa.

4. Protocol conversion in the oneM2M-based IoT system for PHDs

4.1. Protocol conversion process

As mentioned previously, in our earlier study [18] we performed the protocol conversion process for oximetry data only. This study extends the protocol conversion process for various biomedical data, including ECG, glucose, and blood pressure. There are several types of ISO/IEEE 11073 communication messages: Association Request/Response messages, Present Request/Response messages for notice configuration, Present Request/Response messages for sensing data storage, and Association Release messages.

- Association Request/Response message: When a PHD (ADN-AE) wants to connect to a gateway (MN-CSE) for communication, the PHD sends an Association Request message to the gateway, in order to establish a connection between the PHD and the gateway. Upon receiving the message, the gateway finds the ID of the PHD in the message and checks if the PHD ID exists in its resource tree. The existence of the PHD ID indicates that the PHD has been registered in the gateway, and an Association Response message is sent to the PHD.
- Present Request/Response message for notice configuration: When a PHD ID does not exist in the resource tree, the PHD must send a Present Request message to the gateway; this message contains information such as the PHD's environment configuration and biomedical data format. Upon receiving the message, the gateway stores the received information in its resource tree and sends a Present Response message to the PHD.

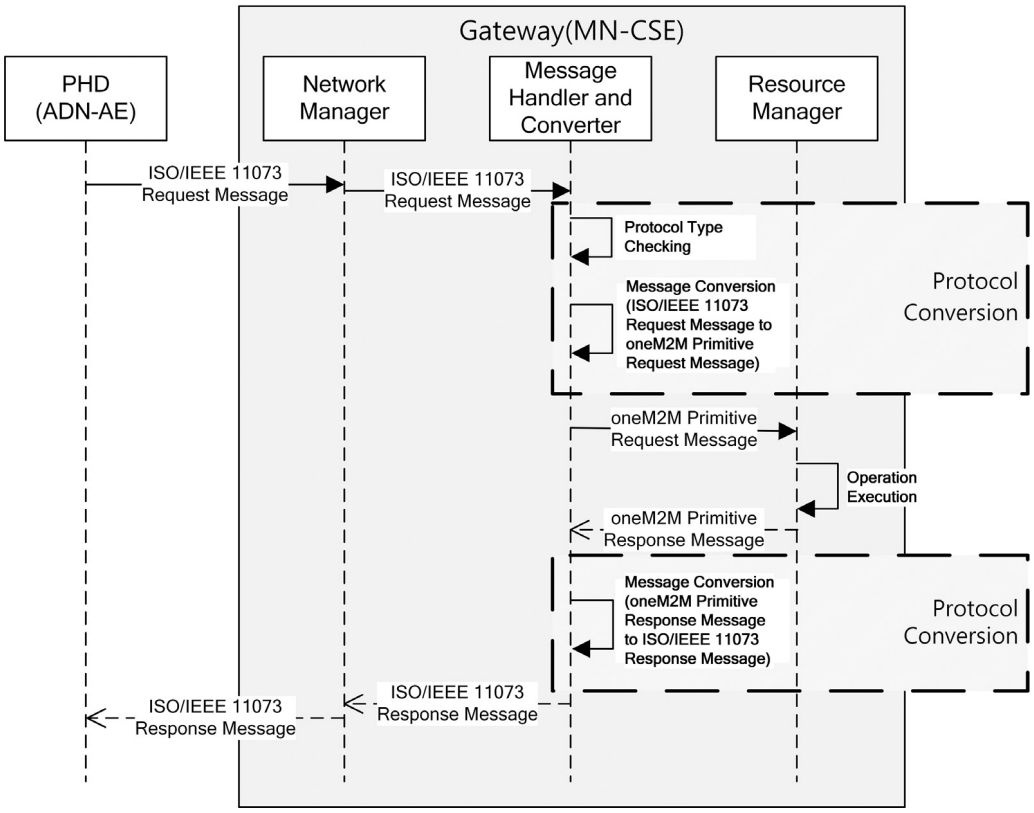

Fig. 3. Communication message flow between a PHD and a gateway.

```
switch (APDU Choice Type) {
  case "ISO/IEEE 11073 Association Request message" : {
    check if "Data Protocol ID" is correct and if "System Type" is a PHD
    get PHD ID from "System ID" for ADN-AE ID
    check if the PHD with the PHD ID has been registered in the resource tree of
      the gateway
    if not registered, have the PHD send an ISO/IEEE 11073 Present (Notice
      Configuration) Request message for registration
    build the header part of the resulting oneM2M Retrieve Request message
      represented as an XML document
    put "<operation> " and "<to> " tags into the resulting oneM2M Retrieve
      Request message
    build the remaining part of the resulting oneM2M Retrieve Request message
      including "request identifier" and "resource type"
    store other parts of the ISO/IEEE 11073 Association Request message for
      building the corresponding ISO/IEEE 11073 Association Response
      message later
  }
```

Fig. 4. Algorithm for converting ISO/IEEE 11073 Association Request messages into oneM2M Retrieve Request messages.

- Present Request/Response message for sensing data storage: After a communication connection is established between the PHD and the gateway, data sensed by the PHD can be transferred to the gateway via a Present Request message; this allows the sensing data to be stored. A Present Response message is sent by the gateway to the PHD, to indicate whether the sensing data was stored successfully.
- Association Release message: An Association Release message is sent when the PHD or the gateway wants to release the communication connection.

Fig. 3 shows communication message flow between a PHD (AND-AE) and a gateway (MN-CSE) in the protocol conversion process.

- From a PHD, a gateway that is responsible for managing the PHD receives the association-related message (i.e., Association request/release message) or present-related message (i.e., Present Request message for notice configuration/sensing data storage) in the ISO/IEEE 11073 format.
- There are three program modules in the gateway—a Network Manager module, a Message Handler and Protocol Converter module, and a Resource Manager module. Upon receiving the message from the PHD, the Network Manager module in the gateway delivers the message to the Message Handler and Protocol Converter module for message analysis and protocol conversion.
- The Message Handler and Protocol Converter module determines the protocol type used to transmit the delivered message.

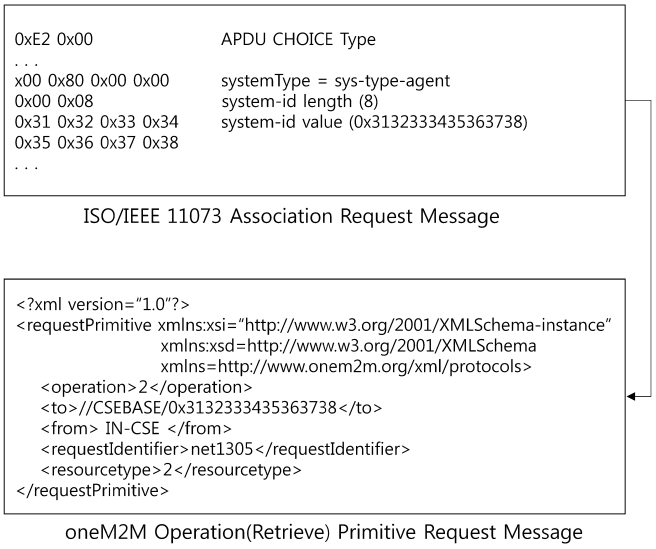

Fig. 5. Conversion of an Association Request message (ISO/IEEE 11073) into a Retrieve Request message (oneM2M).

If the ISO/IEEE 11073 protocol is used, the module extracts the necessary information from the message to construct the corresponding oneM2M Primitive Request message(s), which are delivered to the Resource Manager module. Other information in the delivered message is stored in the gateway, to be used when the related Response message is sent back to the PHD.

- The Resource Manager module is responsible for managing the resource tree that stores the necessary information for oneM2M messages. Upon receiving the oneM2M Primitive message(s) from the Message Handler and Protocol Converter module, the Resource Manager module executes the necessary operations according to the oneM2M message. Then, the oneM2M Primitive Response message(s) is (are) constructed to deliver the operation results/status to the Message Handler and Protocol Converter module.
- Upon receiving the oneM2M Primitive Response message(s) from the Resource Manager module, the Message Handler and Protocol Converter module converts the message(s) into the corresponding ISO/IEEE 11073 Response message, in order to deliver it (them) to the Network Manager module.
- Finally, the Network Manager module sends the ISO/IEEE 11073 Response message to the PHD.

4.2. ISO/IEEE 11073 Association Request message conversion

An algorithm for converting ISO/IEEE 11073 Association Request messages into oneM2M Retrieve Request messages is shown in Fig. 4.

The Association Request message is sent when a PHD (ADN-AE) wants to establish a connection to the server (IN-CSE). After the Network Manager in a gateway (MN-CSE) receives an Association Request message from a PHD, the Message Handler and Protocol Converter analyze the ISO/IEEE 11073 message and convert it into an appropriate oneM2M message. First, the gateway checks whether "APDU Choice Type" indicates an Association Request message and whether "System Type" indicates an agent (PHD). Using the ADN-AE ID of the PHD in "System ID", the gateway can check whether the PHD has been registered in the oneM2M IoT system.

The Message Handler and Protocol Converter deliver a request message to the Resource Manager, to ask if the PHD (represented by "System ID") has been registered in the resource tree of the gateway. If not registered, the gateway sends the PHD a response message that asks the PHD to send an ISO/IEEE 11073 Present (Notice Configuration) Request message for registration.

If registered, the resource tree of the gateway should contain the ADN-AE ID as the PHD ID. Then, the ADN-AE ID is assigned to the "<to>" tag in the resulting oneM2M message, which is in the form of an XML document. Other data in the Association Request message are reference data for the message conversion, and hence are not contained in the resulting oneM2M message. However, they are stored and used when the related Association Response message is generated to send to the PHD. The process that converts the related Retrieve Response message into an Association Response message is similar and omitted here.

Fig. 5 shows an example of the process for converting an Association Request message into a Retrieve Request message.

```
switch (APDU Choice Type) {
  case "ISO/IEEE 11073 Present Request message" : {
    check if "Event Type" is "MDC_NOTI_CONFIG" for Notice Configuration
    build the header part of the resulting oneM2M Retrieve Request message
      represented as an XML document
    for each "Object Handle" in the ISO/IEEE 11073 Present Request message {
      put a "<container resource ID>" tag and the related information into the
        resulting oneM2M Create Request message
      add the "<container>" node to the resource tree of the gateway
      for each "Attribute ID" {
        put a "<contentInstance resource ID>" tag and the related information
          into the resulting oneM2M Create Request message
        add the "<contentInstance>" node under the "<container>" node in the
          resource tree of the gateway
        for each "Attribute ID" of "MDC_ATTR_ATTRIBUTE_VAL_MAP {
          put a "<container resource ID>" tag and the related information
            into the resulting oneM2M Create Request message
          add the "<container>" node to the resource tree of the gateway
    }}}
    store other parts of the ISO/IEEE 11073 Present (Notice Configuration)
      Request message for building the corresponding ISO/IEEE 11073 Present
      (Notice Configuration) Response message later
  }
```

Fig. 6. Algorithm for converting ISO/IEEE 11073 Present (Notice Configuration) Request messages into oneM2M Create Request messages.

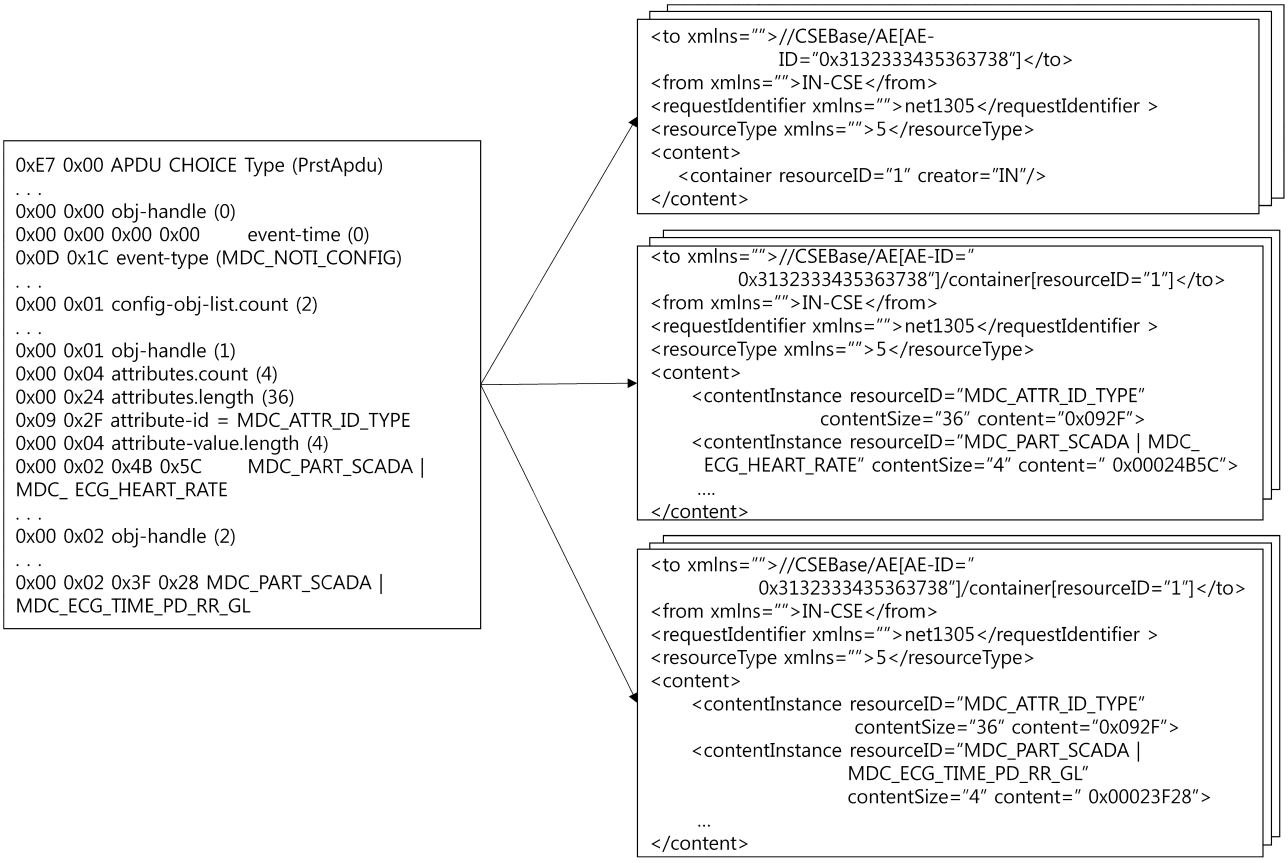

Fig. 7. Conversion of a Present (Notice Configuration) Request message (ISO/IEEE 11073) into Create Request messages (oneM2M).

switch (APDU Choice Type) {
case "ISO/IEEE 11073 Present Request message" : {
check if "Event Type" is "MDC_NOTI_SCAN_REPORT_FIXED" for Store Sensing Data
build the header part of the resulting oneM2M Retrieve Request message represented as an XML document
for each "Scan Report Information Observed Value Object Handle" in the ISO/IEEE 11073 Present Request message {
put a "<contentInstance>" tag and the related information into the resulting oneM2M Update Request message
add the "<content>" node under the "<contentInstance>" node in the resource tree to insert the observed value
add the "<AbsTimeStamp>" tag under the "<contentInstance>" tag to insert the observed time
}
store other parts of the ISO/IEEE 11073 Present (Store Sensing Data) Request message for building the corresponding ISO/IEEE 11073 Present (Store Sensing Data) Response message later
}

Fig. 8. Algorithm for converting ISO/IEEE 11073 Present (Store Sensing Data) Request messages into oneM2M Retrieve Request messages and Update Request messages.

Basic ECG (1- to 3-lead ECG) biomedical data are used in this example. An Association Request message and the resulting Retrieve Request message are depicted in the upper and lower parts of the figure, respectively. In this figure, "APDU Choice Type (0xE200)" indicates that this is an Association Request message and "System Type (0x00800000)" indicates that the requesting PHD is an agent. The "system-id" (0x3132333435363738) in the Association Request message is assigned to the "<to>" tag in the resulting oneM2M message. The URL of the PHD is stored in the "<to>" tag because the PHD is stored in the resource tree of the gateway. "<operation> 2" indicates that this is a Retrieve Request message.

4.3. ISO/IEEE 11073 Present (Notice Configuration) Request message conversion

Fig. 6 shows an algorithm for converting ISO/IEEE 11073 Present (Notice Configuration) Request messages into oneM2M Create Request messages.

Upon receiving an Association Response message from a gateway where a PHD has sent an Association Request message, the

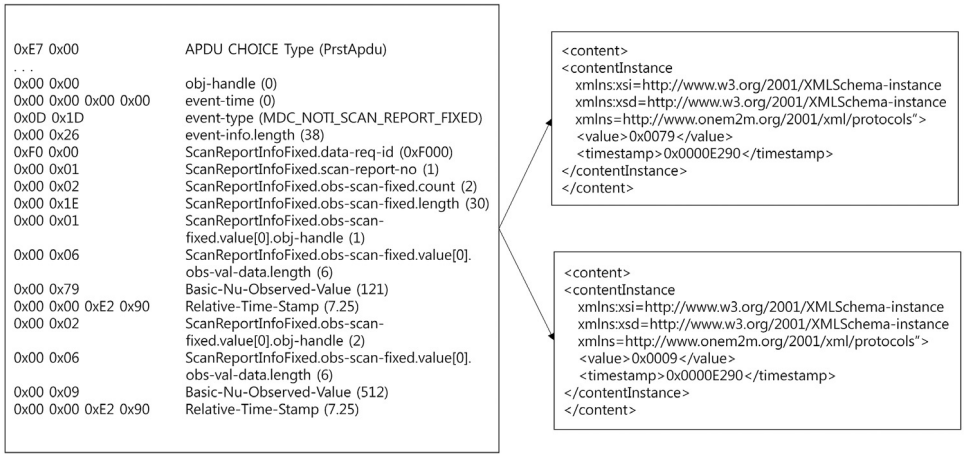

Fig. 9. Conversion of a Present (Store Sensing Data) Request message (ISO/IEEE 11073) into Update Request messages (oneM2M).

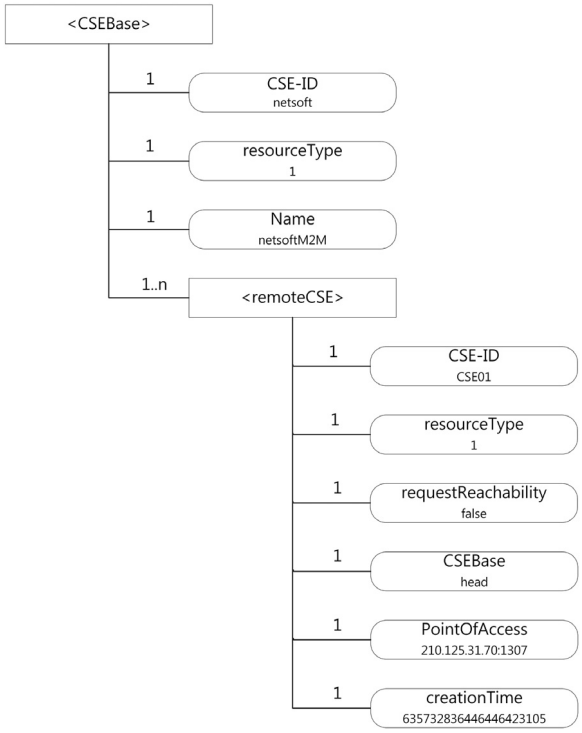

Fig. 10. Initial resource tree in the PHD management server.

PHD checks whether it has been registered in the oneM2M system with which it wants to connect. In the case of nonregistration, the PHD sends an ISO/IEEE 11073 Present (Notice Configuration) Request message to the gateway, to provide the environmental configuration (including the data types) of its biomedical data.

The gateway checks whether "APDU Choice Type" indicates a Present Request message and whether "Event Type" indicates that this message is for Notice Configuration. Then, for each

Table 1
MDS object attributes.

Attribute name	Value
Handle	0
System-Type	Attribute not present
System-Type-Spec-List	Specialization value: {MDC_DEV_SPEC_PROFILE_ECG, 1} and Profile value: {MDC_DEV_SUB_SPEC_PROFILE_ECG, 1} or {MDC_DEV_SUB_SPEC_ PROFILE_HR, 1}
System-Model	{"Manufacturer", "Model"}
System-Id	Extended unique identifier (64-bits) (EUI-64)
Dev-Configuration-Id	Standard config: 0x0258 (600) Extended configs: 0x4000-0x7FFF

"Object Handle", a oneM2M Create Request message is generated. A "<container resource ID>" tag and related information are generated for every "Object Handle" (MDS object) for the resulting oneM2M Create Request message. In addition, a "<contentInstance resource ID>" tag and related information are generated for every "Attribute ID".

Other data in the Association Request message are reference data for the message conversion, and hence are not contained in the resulting oneM2M message. However, they are stored and used when the related Association Response message is generated to send to the PHD. The process that converts the related Retrieve Response message into an Association Response message is similar and omitted here.

Fig. 7 shows an example of the process for converting a Present (Notice Configuration) Request message into Create Request messages for ECG data. A Present (Notice Configuration) message and the resulting Create Request messages are depicted in the left and right portions of the figure, respectively. "APDU Choice Type (0xE700)" indicates that this is a Present Request message. "event-type (0x0D1C)" indicates that this is a Notice Configuration message. "config-obj-list.count (0x0002)" indicates that there are two types of objects (biomedical data) sent by the PHD. A Create Request message is generated for each object. The data in the Present (Notice Configuration) message

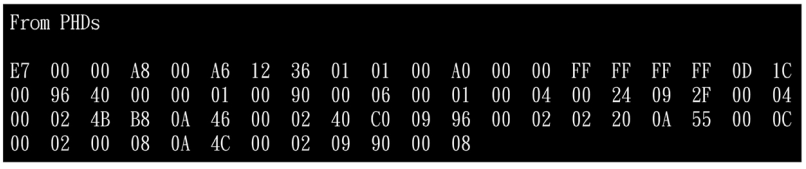

Fig. 11. ISO/IEEE 11073 Present (Notice Configuration) Request message sent by a Pulse Oximetry PHD.

```
Resource Tree로 oneM2M Primitive Response Message
<?xml version="1.0" encoding="utf-16"?>
<responsePrimitive xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.or
g/2001/XMLSchema" xmlns="http://www.onem2m.org/xml/protocols">
  <responseStatusCode xmlns="">2001</responseStatusCode>
  <requestIdentifier xmlns="">net1305</requestIdentifier>
  <to xmlns="">Hwigum</to>
  <from xmlns="">//oneM2M:CSEBase/AE[@AE-ID="1122334455667704"]/container[@resourceID="1"]/container
[@resourceID="MDC_ATTR_ATTRIBUTE_VAL_MAP"]</from>
</responsePrimitive>


Resource Treefh oneM2M Primitive Request Message
<?xml version="1.0"?>
<requestPrimitive xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org
/2001/XMLSchema" xmlns="http://www.onem2m.org/xml/protocols">
  <operation xmlns="">1</operation>
  <to xmlns="">//oneM2M:CSEBase/AE[@AE-ID="1122334455667704"]/container[@resourceID="1"]/container[@
resourceID="MDC_ATTR_ATTRIBUTE_VAL_MAP"]</to>
  <from xmlns="">Hwigum</from>
  <requestIdentifier xmlns="">net1305</requestIdentifier>
  <resourcetype xmlns="">4</resourcetype>
  <content xmlns="">
    <contentInstance xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.
org/2001/XMLSchema" xmlns="http://www.onem2m.org/xml/protocols">
      <resourceID>MDC_ATTR_TIME_STAMP_ABS</resourceID>
      <contentSize>8</contentSize>
      <content>Default</content>
    </contentInstance>
  </content>
</requestPrimitive>


Resource Tree로 oneM2M Primitive Response Message
<?xml version="1.0" encoding="utf-16"?>
```

Fig. 12. oneM2M Operation (Create) Primitive Request message related to the ISO/IEEE 11073 Present (Notice Configuration) Request messages in Fig. 11.

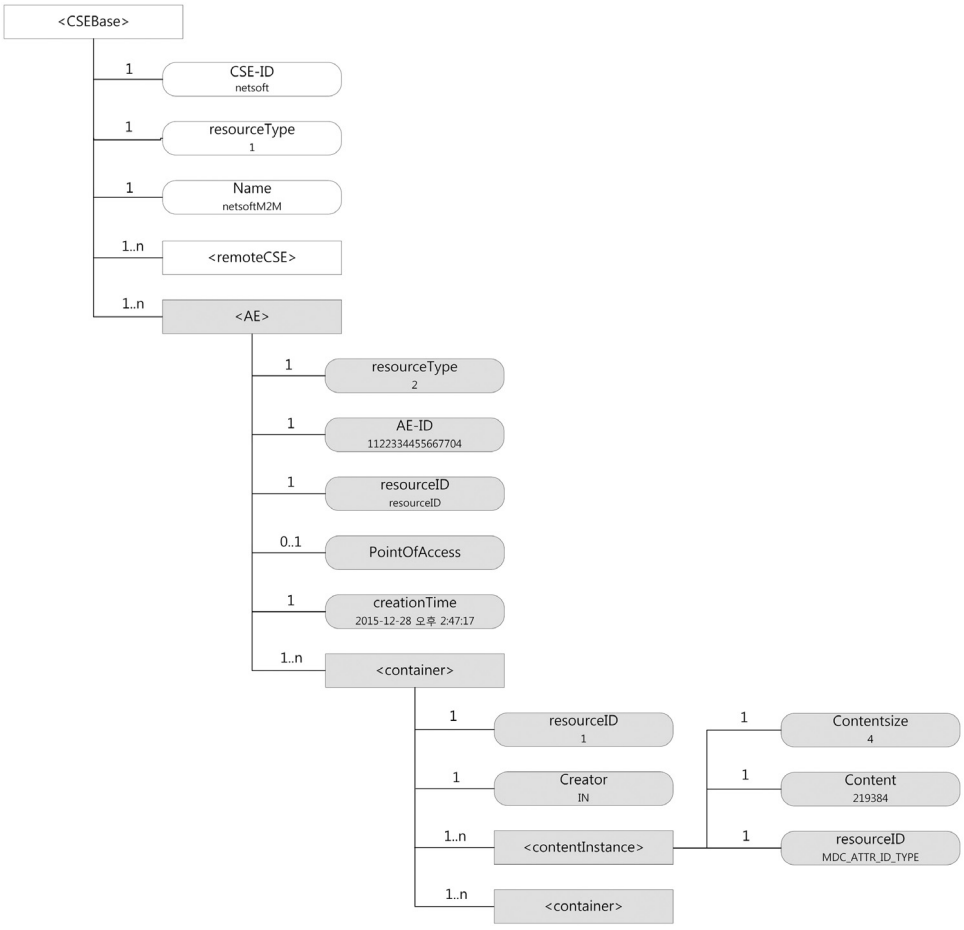

Fig. 13. Updated resource tree after the Pulse Oximetry PHD is registered.

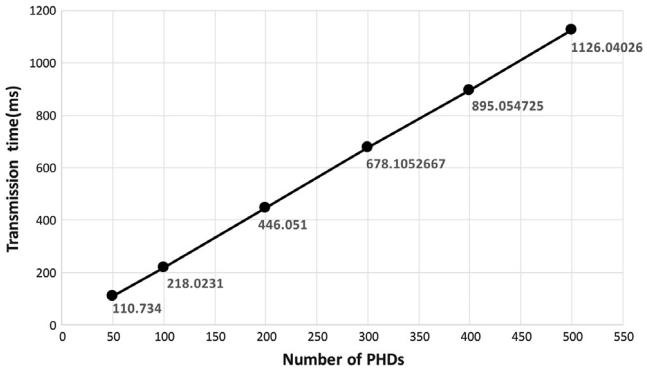

Fig. 14. Average transmission times for messages sent from PHDs to the PHD management server.

represents the detailed description of the object (ECG in this figure) and is assigned to the "<container>" tag in the resulting Create Request oneM2M message. The Create Request message is then delivered to the Resource Manager in order to add the "<container>" node to its resource tree; this indicates that the PHD and its ECG data type are registered. In this figure, two types of measurement data objects are displayed: one for average heart rates (MDC_ECG_HEART_RATE), and one for the R-R interval (MDC_ECG_TIME_PD_RR_GL).

Other mandatory object attributes for basic ECGs (1- to 3-lead ECGs) and MDSs (medical device systems) are shown in Table 1. "System-Type-Spec-List" is a list of device type/version pairs. For a basic ECG agent, a specialization value of MDC_DEV_SPEC_PROFILE_ECG is included in the System-Type-Spec-List attribute. The profile value for a basic ECG agent supporting a simple ECG profile is set to MDC_DEV_SUB_SPEC_PROFILE_ECG.

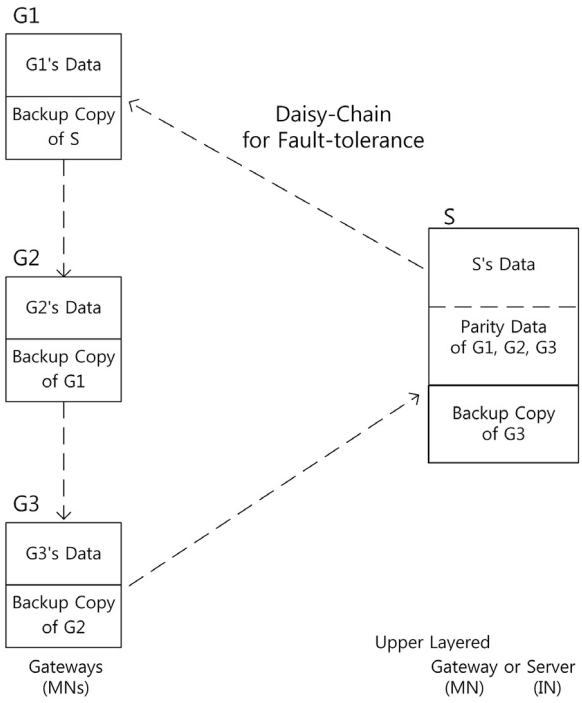

Fig. 16. Baseline structure of the daisy chain for fault-tolerance.

4.4. ISO/IEEE 11073 Present (Store Sensing Data) Request message conversion

Fig. 8 shows an algorithm for converting ISO/IEEE 11073 Present (Store Sensing Data) Request messages into oneM2M Retrieve Request messages and Update Request messages.

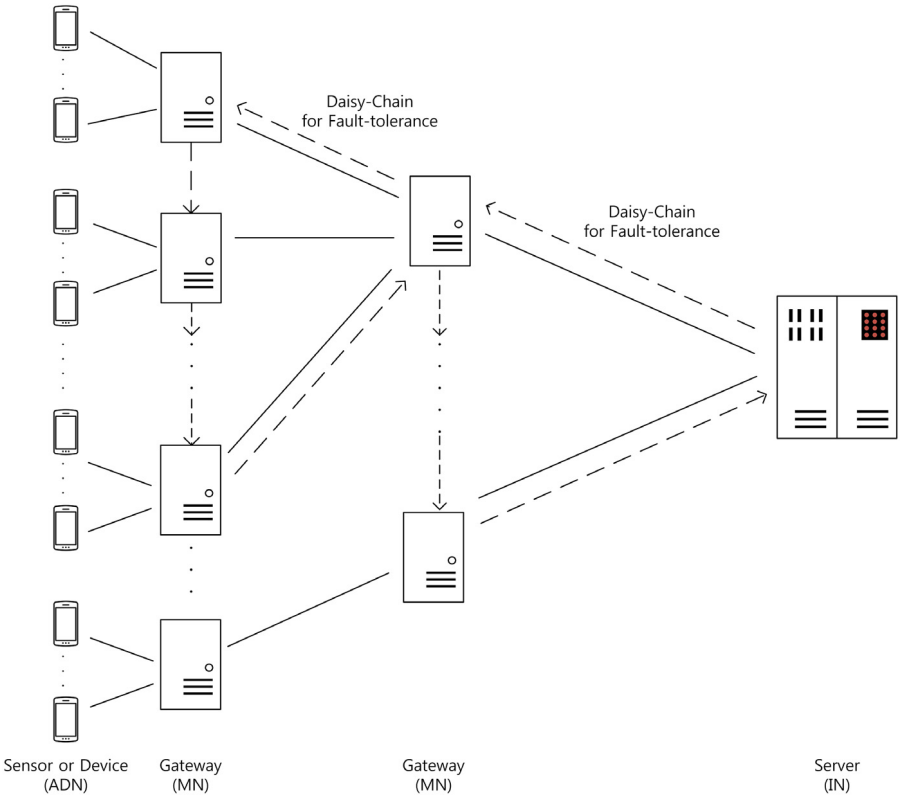

Fig. 15. Augmented multilayered oneM2M-based IoT system model.

For every gateway G in the daisy-chain:
Whenever G's data is updated or a certain event occurs:
① G sends the backup copy of the data to POST(G) in the daisy chain
② G sends the backup copy of the data to the upper gateway or to server S in order to recalculate the parity data by the upper gateway or S

Fig. 17. Fault-tolerance algorithm: backup phase.

(Case 1)
For a fault on a single gateway G:
① G requests S that G's backup copy is sent to G
② S requests that POST(G) send its data to G
③ G receives the backup copy from POST(G)

(Case 2)
For faults on two nonconsecutive gateways in the daisy chain:
Perform the same operations as Case 1 for every faulty gateway

(Case 3)
For faults on two consecutive gateways Gi and Gj in the daisy-chain:
(Case 3-1)
If both Gi and Gj are gateways and Gi is PREV(Gj):
① Both Gi and Gj request S that their backup copies send to them
② S requests that POST(Gj) send Gj's backup copy to Gj
③ Gj recovers its data from the received backup copy
④ S recovers Gi's data by using all data in the daisy chain, including S's parity data; S then sends Gi's recovered data to Gi
⑤ S requests that PREV(Gi) and PREV(Gj) send their data to Gi and Gj, respectively, for their backup copies
⑥ Finally, Gi and Gj restore the backup copies of PREV(Gi) and PREV(Gj), respectively

(Case 3-2)
If Gi is the upper-layered gateway and Gi is PREV(Gj):
① Let Gi recover its data as well as parity data by using the daisy chain at the upper layer
② Gi requests the backup copy of Gj be sent from POST(Gj)
③ Gj recovers its data from the received backup copy
④ Gi sends its data to Gj, and Gj restores the backup copy of Gi
⑤ Gi requests that PREV(Gi) send its data to Gi
⑥ Finally, Gi restores the backup copy of PREV(Gi)

(Case 3-3)
If Gj is the upper-layered gateway and Gi is PREV(Gj):
① Let Gj recover its data as well as parity data by using the daisy chain at the upper layer
② Gj recovers Gi's data by using all data in the daisy chain, including Gj's parity data, and sends Gi's recovered data to Gi
③ Gj restores the Gi's recovered data as the backup copy of Gi
④ Gj requests that PREV(Gi) send its data to Gi
⑤ Finally, Gi restores the backup copy of PREV(Gi)

Fig. 18. Fault-tolerance algorithm: fault recovery phase.

When a PHD measures biomedical data and wants to store it in the server, the PHD sends the gateway an ISO/IEEE 11073 Present (Store Sensing Data) Request message. Fig. 9 shows of the process for converting Present (Store Sensing Data) Request messages into Retrieve Request messages and Update Request messages. When a PHD wants to send sensed ECG data to a gateway in a oneM2M system, a Present (Store Sensing Data) Request message is sent. A Present (Store Sensing Data) message and the resulting Update Request messages are depicted in the upper and lower portions of the figure, respectively. "APDU Choice Type (0xE700)" and "Event Type (0x0D1D)" indicate that this is a Present (Store Sensing Data) Request message. Two heart beat measurements are delivered; 121 bpm and 512 ticks for 7.25 s.

The algorithms explained in Sections 4.2 and 4.3 are executed once, while the algorithm in Section 4.4 is executed whenever sensing data is stored. Obviously, the complexity of the algorithm in Section 4.2.3 is $O(m)$, where m is the number of sensing data elements stored in one ISO/IEEE 11073 message. Therefore, the conversion process proposed in this paper causes some minor overhead.

5. Experiments

A prototype of the proposed M2M-based IoT system for PHDs was constructed, and evaluated in several experiments. Table 2 shows experiment environments for this study. PCs are used for the PHD management server (IN-CSE) and gateways (MN-CSEs), while notebooks are used for PHDs (ADN-AEs). In the experiments, up to 500 Pulse Oximetry PHDs [11] were used. Because hundreds of actual PHDs cannot be obtained, threads that emulate PHDs were generated and used as ADN-AEs for the experiments.

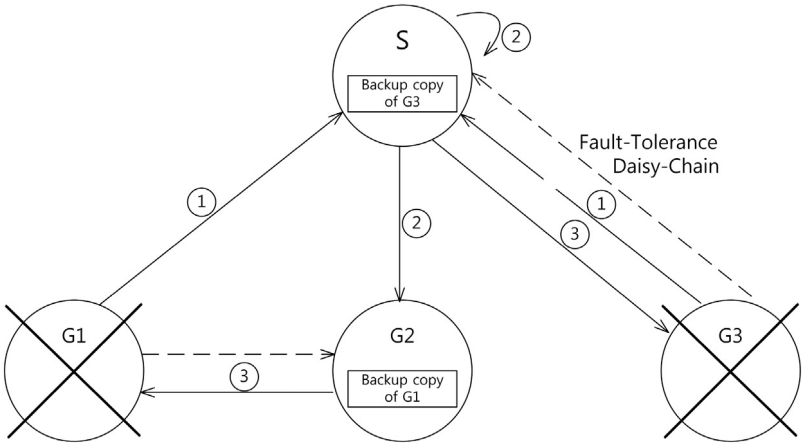

Fig. 19. Fault recovery procedure in the case of two nonconsecutive gateway faults (Case 2).

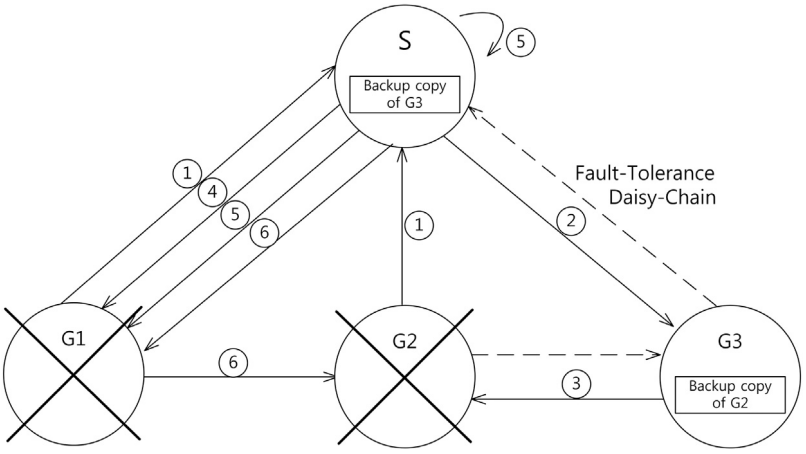

Fig. 20. Fault recovery procedure in the case of two consecutive gateway faults (Case 3-1).

Table 2
Experiment environments.

	PHD management server	Gateway	PHD
CPU	Intel Core i7-3770 (3.4 GHz)	Intel Core i5-650 (3.2 GHz)	Intel Core i3-2367M (1.4 GHz)
Main memory	8 GB	4 GB	2 GB
HDD	SSD	SSD	HDD
Operating system	Windows 7	Windows 7	Windows 7

Fig. 10 shows the initial resource tree in the PHD management server

When a Pulse Oximetry PHD wants to register in the PHD management server, it sends the server an ISO/IEEE 110703 Present (Notice Configuration) Request message, as shown in Fig. 11. Upon receiving the message, the gateway converts the ISO/IEEE 110703 message into an XML document representing the oneM2M Operation (Create) Primitive Request message, as shown in Fig. 12. Finally, the oneM2M message is sent to the server, in order to register the PHD in the server. Registration of the PHD is realized by adding a new (shaded) "<AE>" node in the resource tree of the server, as shown in Fig. 13.

Another experiment analyzed the extent to which protocol conversion overhead affects system performance, especially when a large number of PHDs are being utilized. In the experiments, threads representing up to 500 Pulse Oximetry PHDs were generated to send measured SpO_2 values simultaneously. Fig. 14 shows the average transmission times for messages sent from the PHDs to the PHD management server. The figure shows that the average transmission times increase linearly as the number of PHDs increases; this indicates that the protocol conversion overhead does not seriously affect system performance when many PHDs are involved.

6. Fault-tolerant algorithm for a reliable multilayered oneM2M-based IoT system

6.1. Augmented multilayered oneM2M-based IoT system model

Fig. 15 shows the augmented multilayered oneM2M-based IoT system model used in this study. The multilayered oneM2M-based IoT system (Fig. 1) is augmented with daisy chains for the purpose of fault tolerance in this study. In this figure, the lower-layered gateways are linked together to form a daisy chain for fault tolerance along with the upper-layered gateway (or server) that is responsible for managing the lower gateways. PHDs (ADNs) are not included in the process of fault tolerance because most ADNs in an IoT system do not have sufficient processing power and memory to store backup copies of other ADNs. Moreover, because ADNs do not use information such as resource trees for a oneM2M-based IoT system, they have little information to recover from its failure.

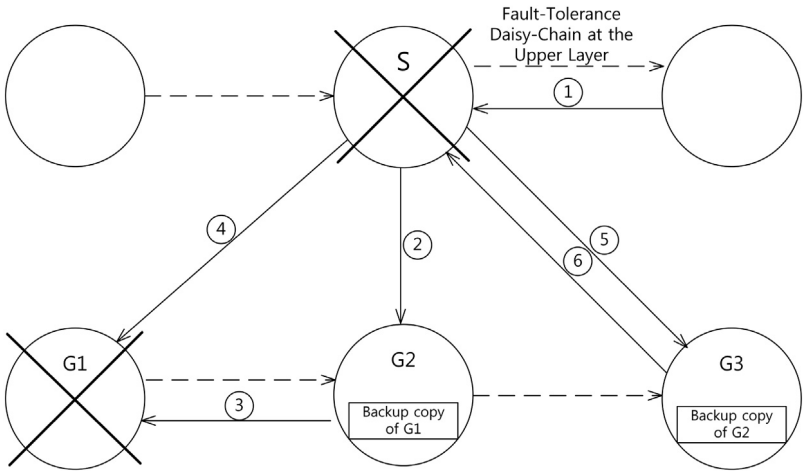

Fig. 21. Fault recovery procedure in the case of two consecutive gateway faults (Case 3-2).

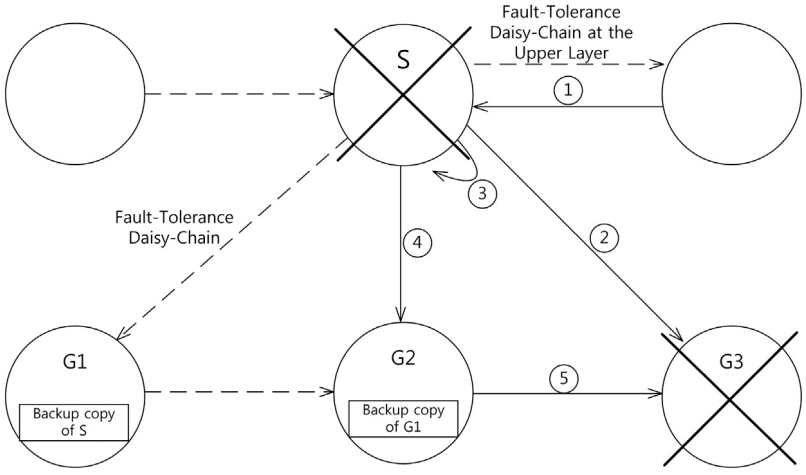

Fig. 22. Fault recovery procedure in the case of two consecutive gateway faults (Case 3-3).

```
<container resourceID="RAID4DAISYinformation" creator="IN">
  <content Instance resourceID="0" contentSize="2" content="IN" />
  <content Instance resourceID="1" contentSize="5" content="CSE01" />
  <content Instance resourceID="2" contentSize="5" content="CSE02" />
  <content Instance resourceID="3" contentSize="5" content="CSE03" />
</container>
</container>
```

Fig. 23. Daisy chain information in the resource tree used in the experiment.

6.2. Fault-tolerant algorithm using daisy chains

Fig. 16 shows the baseline structure of a daisy chain for fault tolerance used in this study. In this figure, three lower-layered gateways (Gs) and one upper-layered gateway, or the server (S) that forms a daisy chain necessary to participate in the fault-tolerance process, exist. G1 is PREV(G2), the previous gateway of G2, whereas G3 is POST(G2), the posterior gateway of G2 in the daisy chain. Every gateway stores a backup copy of its previous gateway in the daisy chain. G1, G2, and G3 store the backup copies of S, G1, and G2, respectively. The upper-layered gateway or the server S stores the backup copy of G3 as well as the parity data of all members of the daisy chain. The parity data are obtained by performing exclusive-OR operations on all data in the daisy chain.

The proposed fault-tolerant algorithm consists of two phases: backup and fault recovery. Figs. 17 and 18 show the backup phase and fault recovery phase algorithms, respectively. In the backup phase, a gateway stores the backup copy sent by the previous gateway in the daisy chain whenever the previous gateway's data are updated or a certain event (e.g., time expiration) occurs. In addition, the parity data stored in S is updated whenever any data in the daisy-chain are updated or a certain event (e.g., time expiration) occurs.

In Fig. 18, three cases are considered for fault recovery. In Case 1, a faulty gateway G requests its data to the upper gateway (or server) after which G is up and running. The upper gateway (or server) then requests that POST(G), the posterior gateway of G, in the daisy chain send to G the backup copy of G that is stored in POST(G).

Fig. 19 illustrates the means by which to recover the faulty gateways when two nonconsecutive faulty gateways (G1 and G3) exist, as in Case 2. The daisy chain alive from the faults is represented by dotted lines. The fault recovery procedure for the case is as follows: ① the faulty gateways G1 and G3 request their data send to them after which G1 and G3 are up and running; ② S requests that the POST(G1) (POST(G3)) in the daisy chain send to G1 (G3) the backup copy of G1 (G3) that is stored in G2 (S); ③ G2 and S then send the backup copies of G1 and G3 to G1 and G3, respectively. Finally, the daisy chain for fault tolerance is fully recovered.

When two nonconsecutive faulty gateways (G1 and G2) exist, as in Case 3, the fault recovery procedure for the case becomes somewhat complicated. In Case 3-1, in which only faulty gateways (i.e., neither faulty server nor faulty upper-layered gateway) are considered, as shown in Fig. 20, the fault recovery procedure for

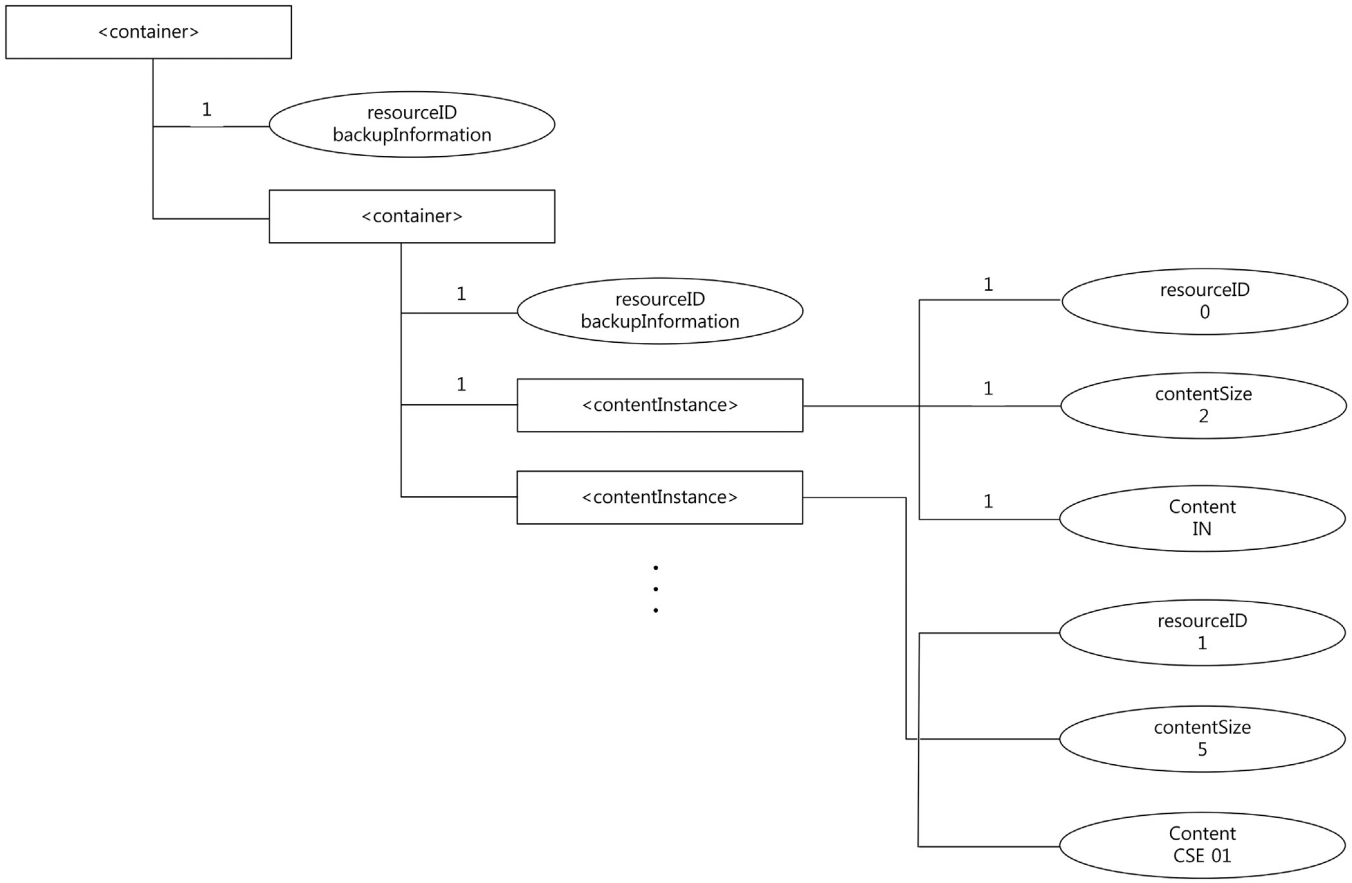

Fig. 24. Resource tree of the daisy chain information used in the experiment.

```
GET /IN/container-recoveryData HTTP/1.1
Host: 210.125.31.70:1305
Accept: application/onem2m-resource+xml
Content-Type: application/onem2m-resource+xml
locale: ko
From: CSE01
X-M2M-RI: net1305
```

Fig. 25. Request for recovery message from CSE01.

the case is as follows: ① the faulty gateways G1 and G2 request their data to the upper gateway (or server) S, after which G1 and G2 are up and running; ② S requests that G3 (= POST(G2)), in the daisy chain send to G2 the backup copy of G2; ③ G2 receives its backup copy from G3, and G2 recovers its data; ④ S recovers G1's data by using G2's data, G3's data, and S's parity data, and then S sends G1's recovered data to G1; ⑤ For backup copies, S requests that G1 (= PREV(G2)) and S (= PREV(G1)) send their data to G2 and G1, respectively; ⑥ G1 and G2 then recover the backup copies of S and G1, respectively. Finally, the daisy chain for fault tolerance is fully recovered.

When two nonconsecutive faulty gateways (S and G1) exist and one of them is the upper-layered gateway (or the server), as in Case 3-2 (shown in Fig. 21), the fault recovery procedure for the case is as follows: ① Let S recover its data as well as parity data by using the daisy chain at the upper layer; ② S requests that G2 (= POST(G1)) in the daisy chain send G1 the backup copy of G1; ③ G2 sends G1 the backup copy of G1, and G1 recovers its data; ④ S sends its data to G1, and G1 recovers the backup copy of S; ⑤ S requests that G3 (= PREV(S)) send G3's data to S; ⑥ S then restores the backup copies of G3. Finally, the daisy chain for fault tolerance is fully recovered. The explanation for Case 3-3 in Fig. 22 is similar and thus omitted here.

6.3. Correctness of the fault-tolerant algorithm using daisy chains

Let us consider the correctness of the proposed algorithm explained in the previous section.

(Case 1): It is straightforward in the sense that, because G's backup data is stored in POST(G), it is enough for G to receive G's backup data from POST(G).

(Case 2): Because there are two nonconsecutive faulty gateways G1 and G2, POST(G1) and POST(G2) are alive and G1 (G2) can receive its backup data from POST(G1) (POST(G2)).

(Case 3-1): In this case, Gi (= PREV(Gj)) and Gj are faulty. Because POST(Gj) is alive, Gj can recover its data simply by receiving its backup data from POST(Gi), as in step ③. When faults occurred on Gj, Gi's backup data stored in Gj was also lost. Therefore, Gi cannot recover its data just by receiving its backup data from Gj (= POST(Gi)). Instead, Gi's data can be recovered by performing exclusive operations on all data in the daisy chain, including S's parity data, as in step ④.

(Case 3-2): In this case, Gi, the upper-layered gateway, and Gj (= POST(Gi)) are faulty. Gi cannot recover from any gateways located at the current layer because POST(Gi) is faulty and no gateways have the parity data on all gateways at the current layer. Instead, Gi can recover its data as well as the parity data by applying the proposed algorithm to gateways at the upper layer, as in step ①. After Gi is recovered, only Gj still remains to be recovered, which is similar to Case 1.

(Case 3-3): This case is similar and thus omitted here.

```
HTTP/1.1 200 OK
Content-Length: 1245
Content-Type: application/onem2m-resource+xml;charset=UTF-8
210.125.31.70:1307/CSE02/container-recoveryData/container-RAID4DAISYdata/contentInstance-R4Dbackup
X-M2M-RI: net1305

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<m2m:contentInstance
xmlns:m2m="http://www.onem2m.org/xml/protocols"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<resourceType>4</resourceType>
<resourceID>R4Dbackup</resourceID>
<creationTime>2015-10-16 오후 3:32:11</creationTime>
<lastModifiedTime>2015-10-16 오후 3:32:11</lastModifiedTime>
<contentSize>4182</contentSize>
<content>0x786D6C2076657273696F6E3D22312E30223F3E3C435345426173652078 6D6C6E733A7873693D22687474703A2F2F7777772E77332E6F72672F323030312F584D4C536368656D6122
D6E616D653D226E6574736F66744D324D2220786D6C6E733D22687474703A2F2F7777772E6F6E656D326D2E6F72672F786D6C2F70726F746F636F6C7322207265736
F7572636554797065 3D223122204353452D49443D226E6574736F6674223E3C72656D6F7465435345207265736F7572636554797065 3D223122063726174696F6
E54696D653D22363335373332383336343436343233313035222070 6F696E744F664163636573733D223231302E313235 2E33312E37303A313330362220435345426
173653D22686561642204353452D49443D22435345303122207265717565737452656163686162696C6974793D2266616C73652220786D6C6E733D2222202F3E3C2
F435345426173653E653E</content>
</m2m:contentInstance>
```

Fig. 26. oneM2M response message for recovery.

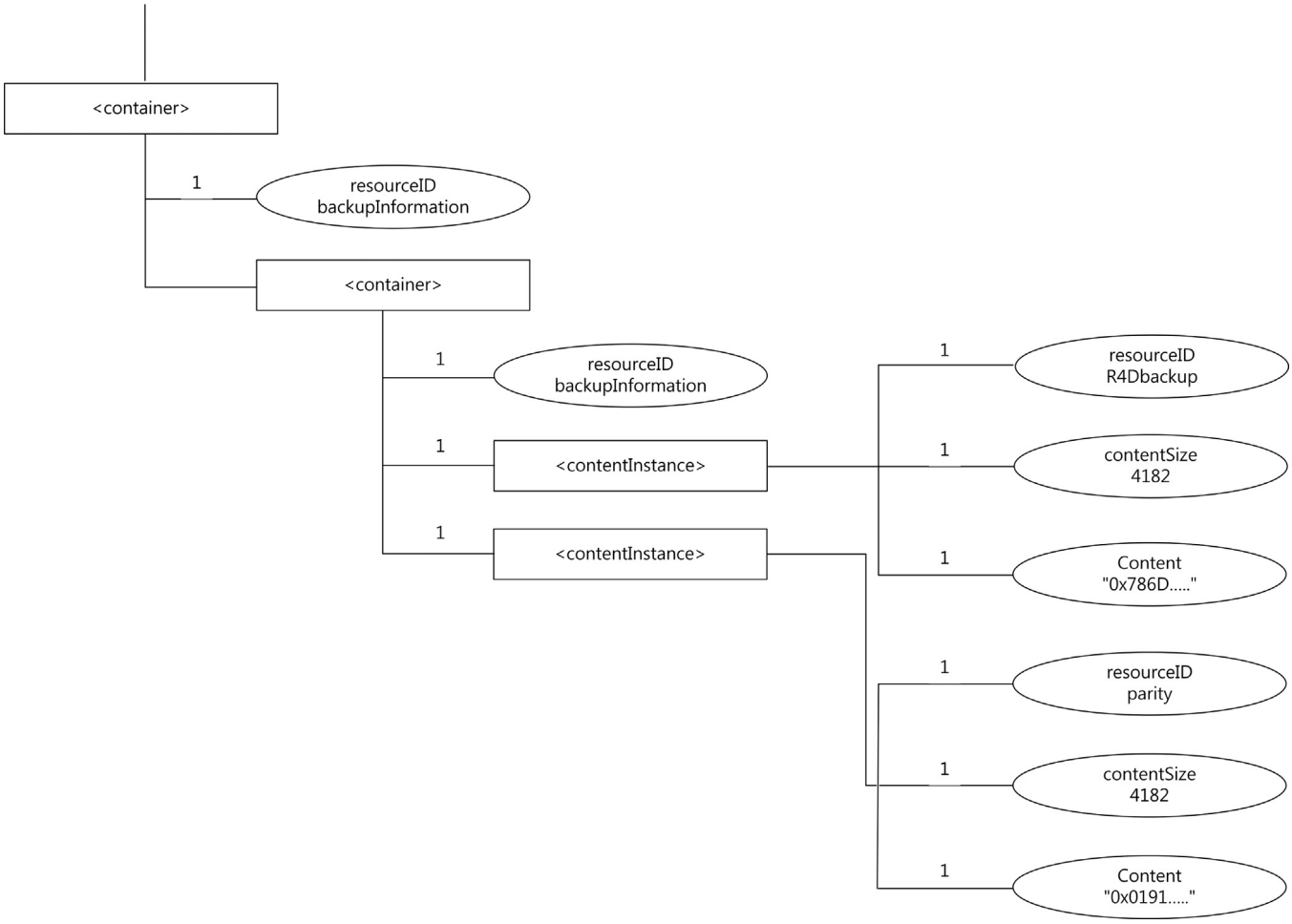

Fig. 27. Backup copy and parity data in the resource tree of IN.

6.4. Experiments on the fault-tolerant algorithm using daisy chains

For this study, the resource trees of the oneM2M-based IoT system were expanded to store information on daisy chains, backup copies, and parity. In this experiment, the expanded resource tree of a gateway is used as data to be recovered after failure of the gateway. Fig. 23 shows the daisy chain information in the resource tree for experiments on the proposed fault-tolerant algorithm. In this figure, there are one server (“IN”) and three gateways (“CSE01”, “CSE02”, and “CSE03”). The resource tree can be depicted as shown in Fig. 24.

When CSE01 is up after its failure, CSE01 requests IN for its data for recovery, as shown in Fig. 25. Then, IN requests CSE02 to send CSE01’s backup data stored in CSE02. Fig. 26 shows a oneM2M response message to send CSE01 the backup data for recovery. The box in the figure shows the backup data of 4182 bytes to be sent

for CSE01's recovery. In this experiment, the backup data is the expanded resource tree of CSE01. The server IN has a backup copy of CSE03's data and parity data of all members of the daisy chain. The resource tree in server IN is depicted in Fig. 27.

7. Conclusion

In this paper, a reliable oneM2M-based IoT system for PHDs is proposed. (Programs on) PHDs are used as ADN-AEs in the oneM2M system. Gateways (MN-CSEs) are used to link PHDs and the PHD management server (IN-CSE). One of the important tasks assigned to the gateways is to convert the ISO/IEEE 11073 protocol messages from PHDs into the IoT server's oneM2M protocol messages, and vice versa. Communication message flow between PHDs and gateways is proposed for managing the protocol conversion process, and the relationships between oneM2M protocol messages and ISO/IEEE 11073 protocol messages are described. The oneM2M-based IoT system for PHDs was constructed, and evaluated in various experiments. The experiments show that the protocol conversion works effectively, and that the system does not suffer serious performance degradation from the conversion process, even when the number of PHDs is quite large.

We also proposed a fault-tolerant algorithm for a reliable IoT system in which gateways on the same layer in the system are linked to form a daisy chain for fault tolerance at the level, and a gateway stores the backup copy of the previous gateway positioned immediately ahead of the gateway in the daisy chain. The backup copy of the last gateway in the daisy-chain is stored by the upper-layered gateway (or server) in the system. The upper-layered gateway stores the parity data of the daisy chain as well. In this manner, as many as two gateway faults occurred at the same time can be recovered. The resource trees of the oneM2M-based IoT system were expanded to store information on the daisy chains, backup copies, and parity for this study. We evaluated our proposed fault-tolerant algorithm using the daisy chain in experiments on the multilayered oneM2M-based IoT system. Our experiments reveal that the proposed algorithm can recover from faults on gateways in the oneM2M-based IoT system.

Acknowledgment

This research was supported by the Basic Science Research Programs through the National Research Foundation of Korea (NRF), funded by the Ministry of Education, Science and Technology (No. NRF-2015R1D1A3A03019278).

References

[1] ITU, The Internet of things, 2005.
[2] European Commission, Internet of things-an action plan for Europe, 2009.
[3] CISCO, How the Internet of everything will change the World, 2012.
[4] L.A. Grieco, M. Alaya, T. Montei, K. Drira, Architecting information centric ETSI-M2M systems, in: PERCOM (Pervasive Computing and Communications) Workshops, 2014, pp. 211–214.
[5] oneM2M, Functional architecture (TS-0001-V1.6.1), http://www.onem2m.org, 2015.
[6] oneM2M, Requirements (TS-0002-V1.0.1), http://www.onem2m.org, 2015.
[7] Gheorghe Sebestyen, Anca Hangan, Stefan Oniga, Zoltan Gal, eHealth solutions in the context of Internet of things, in: IEEE International Conference on Automation, Quality and Testing, Robotics, 2014, http://dx.doi.org/10.1109/AQTR.2014.6857876.
[8] Amir-Mohammad Rahmani, Nanda Kumar Thanigaivelan, Tuan Nguyen Gia, Jose Granados, Behailu Negash, Pasi Liljeberg, Hannu Tenhunen, Smart e-Health gateway: Bringing intelligence to Internet-of-things based ubiquitous healthcare systems, in: IEEE Consumer Communications and Networking Conference, 2015. http://dx.doi.org/10.1109/CCNC.2015.7158084.
[9] Health Informatics-Personal Health Device Communication, IEEE Std. 11073-10407 Device Specialization-Blood Pressure Monitor, http://standards.ieee.org, 2014.
[10] Health Informatics-Personal Health Device Communication, IEEE Std. 11073-10417 Device Specialization-Glucose Meter, http://standards.ieee.org, 2014.
[11] KeeHyun Park, SeungHyeon Lim, Message processing at integrated PHD gateways for servicing various PHDs, Int. J. Multimed. Ubiquitous Eng. 9 (3) (2014) 367–374.
[12] KeeHyun Park, SeungHyeon Lim, A secure bio-medical data management system for a very large number of various PHDs, BioMed. Res. Int. 2015 (2015) 17 pages. http://dx.doi.org/10.1155/2015/941053, Article ID 941053.
[13] K. Marzullo, Tolerating failures of continuous-valued sensors, ACM Trans. Comput. Syst. 8 (4) (1990) 284–304.
[14] Deepak Ganesan, Ramesh Govindan, Scott Shenker, Deborah Estrin, Highly-resilient, energy-efficient multipath routing in wireless sensor networks, Mob. Comput. Commun. Rev. 1 (2) (1997) 1–13.
[15] G. Gupta, M. Younis, Fault-tolerant clustering of wireless sensor networks, Wirel. Commun. Sensor Netw. 3 (2003) 1579–1584.
[16] D. Desovski, Y. Liu, B. Cukic, Linear randomized voting algorithm for fault tolerant sensor fusion and the corresponding reliability model, in: IEEE International Symposium on Systems Engineering, 2005, pp. 153–162.
[17] S. Harte, A. Rahman, Fault tolerance in sensor networks using selg-diagnosing sensor nodes, in: IEEE International Workshop on Intelligent Environment, 2005, pp. 7–12.
[18] KeeHyun Park, Joonsuu Park, JongWhi Lee, An IoT system for remote monitoring of patients at home, Appl. Sci. 7 (3) (2017) 260. http://dx.doi.org/10.3390/app7030260.
[19] Zainab H. Ali, hesham A. Ali, Mahmoud M. Badwy, Internet of things (IoT): Definition, challenges and recent research directions, Int. J. Comput. Appl. 128 (1) (2015) 37–47.
[20] Tuan Nguyen gia, Amir M. Rahmani, Tomi Westerlund, Last Hannu Tenhunen, Fault tolerant and scalable IoT-based architecture for health monitoring, in: 2015 IEEE Sensors Applications Symposium, 2015, http://dx.doi.org/10.1109/SAS.2015.7133626.
[21] Sudip Misra, Anshima Gupta, P. Venkata Krishna, Harshit Agarwa, Mohammad S. Obaidat, An adaptive learning approach for fault-tolerant routing in Internet of things, in: 2012 IEEE Wireless Communications and Networking Conference, 2012, pp. 815–819.
[22] S. Chaithra, S. Gowrishankar, Study of secure fault tolerant routing protocol for IoT, Int. J. Sci. Res. 5 (7) (2016) 1833–1838.

Min Woo Woo received his B.Sc. and M.Sc. degrees in Computer Engineering from Keimyung University, Korea in 2014 and 2016, respectively. His interests include IoT System, Fault-tolerant System.

JongWhi Lee received his B.Sc. and M.Sc. degrees in Computer Engineering from Keimyung University, Korea in 2014 and 2016, respectively. His interests include IoT System, Communication Protocols.

KeeHyun Park received his B.Sc. and M.Sc. degrees in Computer Science from Kyungpook National University, Korea, and from KAIST, Korea, in 1979 and 1981, respectively, and his Ph.D. degree in Computer Science from Vanderbilt University, USA, 1990. He has been a professor of Computer Science and Engineering Department at Keimyung University, Korea since March 1981. His research interests include Mobile/Network Communication System, Device Management for u-healthcare Systems, Embedded System and Parallel Processing System.