International Journal on Advanced Science Engineering Information Technology

Vol.8 (2018) No. 2
ISSN: 2088-5334

A Sleep Scheme Based on MQ Broker Using Subscribe/Publish in IoT Network

Wenquan Jin[#1], DoHyeun Kim[#2]

[#] *Computer Engineering Department, Jeju National University, South Korea*
E-mail: [1] wenquan.jin@jejunu.ac.kr, [2] kimdh@jejunu.ac.kr

***Abstract*— Constrained Application Protocol (CoAP) is a transfer protocol that is used for Internet of Things (IoT) devices such as sensors and actuators equip with low power supply, limited computing processor, and constrained network environment. For the constrained devices of CoAP based IoT network, we present a sleepy scheme based on a Message Queue (MQ) broker that supports the subscribe/publish communication architecture in the IoT middleware to enable the devices having better energy consumption. The IoT middleware is a server that providing services for storing and retrieving information of IoT node using Resource Directory (RD) functionality, and supporting the sleepy scheme using MQ broker functionality. The RD provides HTTP services to the HTTP client based application for discovering and looking up information of IoT nodes which are registered to the RD. The functionality of MQ broker is used for performs store-and-forward messaging. The MQ provides the service for IoT nodes to publish data to the middleware. The data shall be subscribed by the client application. The IoT node in the sleep mode that cannot be accessed by the client. Through the IoT middleware, the client can subscribe the IoT node for getting the result from IoT node after wake up. Once the IoT node wakes-up from the sleep status, the IoT middleware publishes the subscribed result to the client.**

***Keywords*— IoT; CoAP; Constrained Application Protocol; resource directory; sleepy; middleware.**

I. INTRODUCTION

A large number of connected device have been deployed to support efficient and comfortable functions through the client applications [1]. The smartphones are the most popular Internet-connected devices which attached our daily life through its applications and sensors to provide heterogeneous services. However, more devices can be deployed and already have been deployed which create worldwide and private networks to support smart and ubiquitous solutions. The Internet of Things (IoT) is comprised by heterogeneous devices to provide greater solutions for the industries of a specific or cross domains [2]. The IoT devices shall be easily configured and automatically do pre-defined jobs such as registration, actuating clone jobs, and sleepy schedules [3]. According to the changes of communication paradigm from human-centric to machine-centric which illustrates the IoT devices do the task without the human touch [4]. Furthermore, the IoT devices need to be describable in order to communicate with other objects or services [5]. For describing the IoT device to the others, the information of IoT device needs to be appeared in the IoT network. In the IoT network, the directory entity is required for storing and providing information of IoT devices such as identifier, name, location, security parameters, and etc [6]. Through the directory entity in the IoT network, clients can look-up the information of IoT devices to access services which are provided by resources of IoT devices.

The service for discovering information in the IoT environment is important to look-up devices [7] [8]. Especially, the IoT devices are deployed in constrained environment with limited power resources, which cannot available to provide services all the time. For the constrained device, the IEFT had proposed the Constrained Application Protocol (CoAP) to reduce the network communication cost through reducing the transmission message size [9]. The CoAP supports a request/response interaction model between application endpoints using REST based service accessing architecture through the UDP [10]. The size of CoAP message is small than HTTP and support more efficient parameter options to build the network entities [11]. For saving the energy of IoT device, the IETF have mentioned the sleepy scheme in IETF-87 [12]. The sleepy feature can be implemented in the MAC layer or external application with the CoAP based application entity of IoT node [13] [14]. The CoAP based IoT node can be used for devices with limited memory and small battery power, and the IoT node is also expected with the sleepy scheme can support better constrained solutions in the CoAP network.

In this paper, we present a sleepy scheme based on the Message Queue (MQ) broker using subscribe/publish in the proposed IoT network where the service provider, IoT middleware, and IoT device are deployed. The middleware includes RD and MQ broker to interact with IoT nodes using the CoAP-based communication. The IoT nodes equip one or more units for sensing and actuator, and are deployed in constrained environment where support the CoAP network. The RD is a server that hosts descriptions of CoAP nodes and shall always allow lookups for retrieving registered information [15]. In CoAP based communication network, the CoAP client can search the CoAP node from the RD. Before the searching process, the CoAP node needs to register its information to the RD [16]. An extension of the RD is MQ broker which enables entities to publish and subscribe information through storing and forwarding messages [17]. The subscribe/publish architecture is used by the communications of wireless, local area network, and remote control, that synchronizes the information in clients with the broker [18]. Once the IoT node goes into a sleep status, then the node sends the information for the current status to the IoT middleware. The client can subscribe the IoT node from the MQ broker of IoT middleware. Once the IoT node goes into a wake-up status, then the client can get the data from the IoT node through the IoT middleware.

Rest of the paper is structured as follows; Section 2 introduces the sleepy scheme in the proposed IoT system. Section 3 introduces the implementation, experiment result of proposed IoT system. Finally, we conclude our paper in Section 4.

II. Material and Method

Figure 1 shows the overall IoT architecture based on the IoT middleware. The IoT architecture includes service provider, IoT middleware, and IoT node. The service provider and IoT middleware communicate through the HTTP communication protocol because most of clients in the Internet are developed using HTTP client. The IoT middleware and IoT node communicate through the CoAP communication protocol because most of devices in the constrained environment. The service provider includes the HTTP client to communicate with the IoT middleware for forwarding request from its clients to the IoT node and managing information of IoT node or other entity in this system. The IoT middleware provides services using REST APIs based on HTTP to the service provider and using REST APIs based on CoAP to the IoT node. The APIs deliver the requests to the handlers of IoT middleware.

The IoT middleware includes the RD for IoT node information managing and MQ brother for forwarding requests from the HTTP client of service provider. The RD provides services to the IoT nodes for registering information to be retrieved by clients. The MQ broker provides services to the client for forwarding requests. In this forwarding process, if the IoT node is in sleep status, therefore, cannot respond the request immediately, then the MQ broker shall store the request and later forward the request message to the destination IoT node. The IoT node in the constrained environment that requests the IoT middleware for notifying the sleep status once it goes into a sleep mode. For applying the sleepy scheme to CoAP-based IoT network, the IoT middleware needs the functionalities of RD and MQ broker. The RD is used to hold the information of IoT node to provide discovery service to the client for looking up the information. The MQ broker enables the IoT node publishes sensing data to the IoT middleware for being subscribed by clients. Based on the RD and MQ, the IoT middleware is used for supporting interoperability with sleepy devices in the CoAP based IoT network.

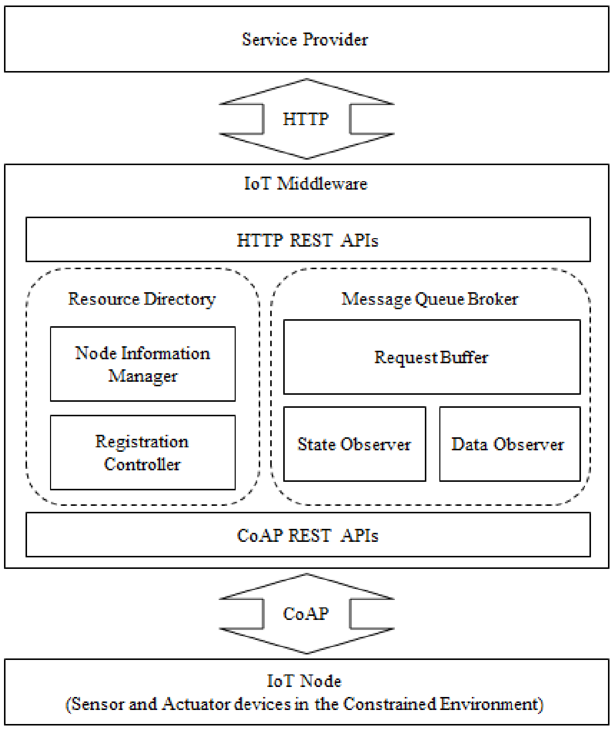


Fig. 1 Overall IoT architecture based on the IoT middleware

Figure 2 shows the sequence diagram for the IoT middleware-based sleepy scheme. The sleepy scheme is used for IoT nodes in the CoAP network. IoT nodes are constrained devices which need to save the energy. Therefore, the devices need the limited energy consumption through the sleepy scheme.

The IoT node can fall into sleep mode through the request from the IoT middleware. The request uses the PUT method with URI coap://{node-uri}/sleepy and query parameters sleep_state and sleep_duration. The URI includes the resource of sleep status which is used for changing the sleep mode of IoT node. The query parameter sleep_state is used for setting up the sleepy mode of IoT node, e.g. 1 for setting up the IoT node go into a sleep status and 0 for setting up the IoT node go into a wake status. The query parameter sleep_duration is used for setting up the sleep duration of IoT node, e.g. the duration time is 5 then the IoT node shall be inaccessible 5 seconds. Once the request message is sent to the IoT node, the information in the IoT node shall be updated and the information in the IoT middleware also shall be updated after receive the response message. The process of updating the IoT node information is handled by the RD of IoT middleware.

Once the IoT node is in the sleep status, the client cannot access the service from the IoT node. The response message shall be indicating the sleep status of destination IoT node. Then the client can request to the IoT middleware to get the sleepy duration value or subscribe the IoT node for getting the response when the IoT node wakes up.

The subscribe/publish process can be done by 5 steps.

Step 1: The client sends the request to the IoT middleware to register the subscription and IoT middleware store the information of request for the client.

Step 2: Once the IoT node becomes wake status after the sat up sleep duration time is passed, the IoT node sends the request to notify the IoT middleware. Then the information of IoT node shall be updated.

Step 3: IoT middleware sends the request to the IoT node that is store for the client.

Step 4: Once the IoT middleware receives the response from the IoT node, then IoT middleware sends the request to the client for publishing the subscribed information by the client.

The subscribe/publish functionality is belong to the MQ broker of IoT middleware. The handlers of request buffer, state observer, and data observer handle the request from clients and IoT nodes for caching, notifying, and updating the information of sleepy and service results.

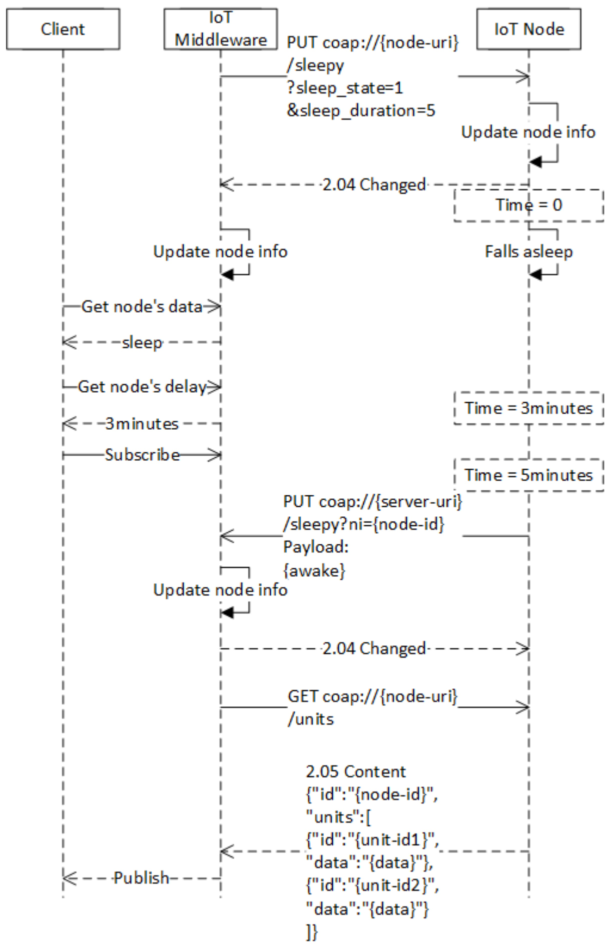


Fig. 2 Sequence diagram for IoT middleware-based sleepy scheme

For presenting the scenario of sleepy scheme with the entities in the CoAP network, we use the business process model to show the interaction of client, IoT middleware, and IoT node in Figure 3.The process begins from the client which sends message to the RD of IoT middleware to get the IoT node's data by accessing service that is provided from the IoT node.

Once the IoT node goes into the sleep status then the IoT middleware returns the sleep information to the client. The IoT middleware checks the IoT node's sleep state by its identifier using RD and hold the request from client using MQ broker until the IoT node becomes wake status. Once the IoT middleware receives the subscription request from the client, then the MQ broker waits the wake-up notification from the IoT node. After the node wakes up from sleep status, then the MQ broker requests to the IoT node based on the request from the client and publish the result of response from the IoT node to the client.

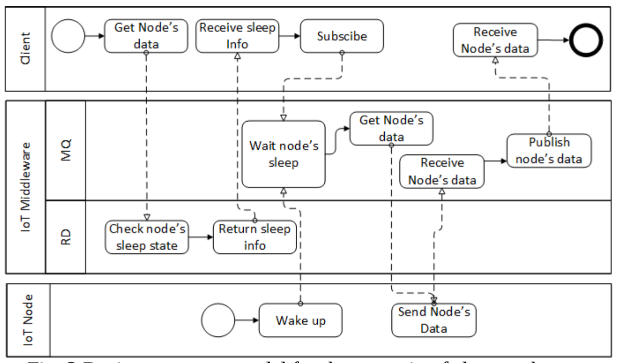


Fig. 3 Business process model for the scenario of sleepy scheme

III. Results and Discussion

The system includes the service provider, IoT middleware, and IoT node for supporting the sleepy scheme to the IoT network where the devices support CoAP based communication. We implement the CoAP based communication for the proposed IoT middleware and IoT node, and the service provider provides HTTP-based web services to its web client. Table 1 shows the development environment for the service provider, IoT middleware, and IoT node. The service provide is developed in C# and the application runs on dot net 4.5 platform with the Windows OS. We use Visual Studio as the development tool to develop the application. The IoT middleware is developed in Java and the application runs on Java Runtime Environment with the Window OS. We use eclipse as the development tool to develop the application. The IoT node is developed in C and the application runs on GCC with the Ubuntu OS. We use gedit as the development tool to develop the application.

The IoT middleware is developed using Java with Californium CoAP framework (Cf) [19]. In order to support HTTP RESTful APIs, we use Apache Tomcat to run the Java servlets of IoT middleware. The IoT middleware includes SQL server for managing data via data access objects. Once the Java servlets run in the Apache Tomcat, there is initialization function for initializing the CoAP resources.

The IoT node is developed using Linux C libraries which are libcoap library and cJSON library for developing the application of IoT node [20] [21]. The library libcoap is used for implementing CoAP resources to support CoAP RESTful APIs, and cJSON is used for parsing transmission messages which are formatted in JSON. The system uses JSON data in the communication between the IoT middleware and IoT node. The library libcoap implements a lightweight application-protocol for IoT devices that are constrained such as computing power, RF range, memory, bandwidth, or network packet sizes. The library is published as open-source software without any warranty. The usage is permitted under the terms of the GNU general public license version 2, higher, or the revised BSD license. In the process of the IoT node, all data is formatted in JSON format. Incoming data and out-going data are formatted in JSON for the interaction between IoT node and IoT middleware.

TABLE I
DEVELOPMENT ENVIRONMENT

Environment	Service Provider	IoT Middleware	IoT Node
OS	Windows 7	Windows 7	Ubuntu 12.4
Runtime environment	.Net 4.5	Java 7	GCC
Tool	Visual Studio	Eclipse	gedit
Language	C#	Java	C

We present the IoT node for constrained environment such as sensor networks and actuator networks using the CoAP communication. The RESTful APIs are supported by the IoT node which enables to provide IoT services by the IoT node.

Figure 4 shows the functional components of IoT node. The node information manager has GET and PUT method handlers for handling IoT node's information. The GET method handler is used for responding the information of IoT node to the requester. The PUT method handler is used for updating the information of IoT node. The sleepy manager is use for updating sleepy information and providing the service for changing the sleepy mode of IoT node. The data manager is use for handling the request to the IoT service that is provided by the IoT node.

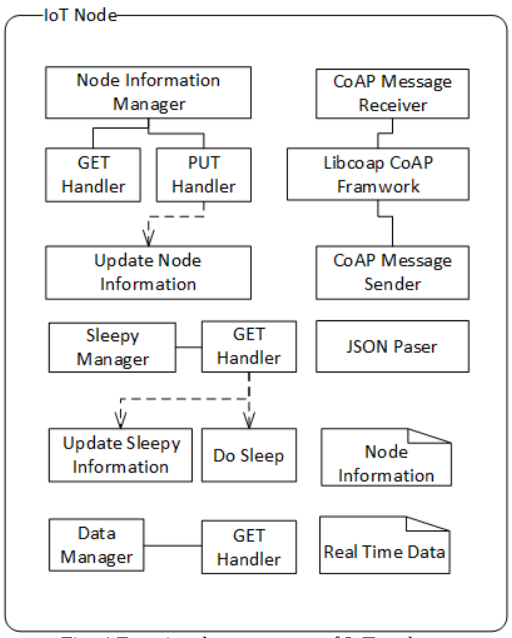


Fig. 4 Functional components of IoT node

The information of IoT node in the IoT node that is stored in the file as the profile of IoT node and the profile is formatted in a JSON data. The profile includes node's information and unit's information which are included in the IoT node application. The version of the node profile is used for synchronizing the information between the IoT node and the IoT middleware. And other attributes are used for functions of the IoT node such as sleep status, sleep duration, notify enable and notify interval. There can be multiple units of a node. The information of the units can be appropriately different without id of the unit. Unit's information includes resource type and interface which are proposed in the RFC 6690. The resource type attribute is an opaque string used to assign an application-specific semantic type to a resource. And the interface description attribute is an opaque string used to provide a name or URI indicating a specific interface definition used to interact with the target resource.

Each IoT node has own URI with IP address. But it is quite common for a node to change its IP address due to rebooting. We design the IoT node seamless synchronize its information with RD in the IoT middleware. Service users of the IoT middleware, which retrieve the IoT node's URI via the ID of node to find the IoT node in the network. For example, a service provider need to send a command to actuate a unit of the IoT node, then the service provider will get the ID which are related the requirement. Service provider requests a command to the IoT middleware with the node ID and unit ID which are parameters of query. And IoT middleware sends the command to the IoT node via node's URI through CoAP with unit ID, the node URI is retrieved using node ID from database. Finally, IoT node receives the command and actuates the unit by the unit ID. Those IDs also are used for retrieving the sleep information of the IoT node. Furthermore, the node and unit of the IoT device also involves own status and features to relate with physical parts. Those parts are represented as data in the IoT network. Therefore, the IDs are the keys to mapping those information.

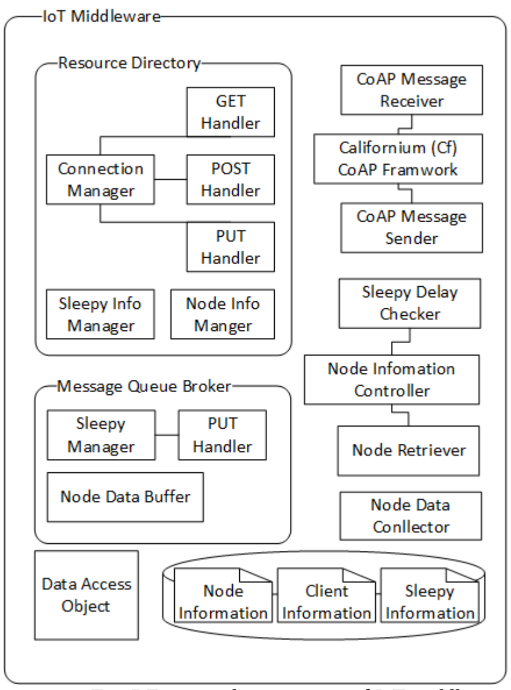


Fig. 5 Functional components of IoT middleware

Figure 5 shows the functional components of IoT middleware. The IoT middleware includes the RD and MQ broker. The RD includes the connection manager to control the registration of IoT node, and manage the sleepy information of IoT node and its information through the sleepy information manager and IoT node information manager. The MQ broker includes the sleepy manager for receiving the request from the IoT node. The sleepy manager is a CoAP resource that has PUT method to handle request from IoT node. In the IoT middleware, we use Cf CoAP Framework to implement CoAP communication for providing CoAP services to the IoT node in constrained network. The HTTP service is use for HTTP client from the service provider.

Figure 6 shows the use case of CoAP resources of IoT middleware. The IoT middleware has 3 CoAP resources for providing CoAP services. The CoAP resource “conn” is use for registering the IoT node information. The resource provides service through GET, POST and PUT method handlers. The GET method handler is use for retrieving the IoT node’s identifier from the database of IoT middleware to get define the unique identifier of IoT node in the system. The POST method handler is use for creating the IoT node information. The PUT method handler is use for updating the IoT node information. The CoAP resource “/observer/data” is use for accepting sensing data from equipment units of IoT node such as sensors and actuators. The CoAP resource “/observer/sleepy” is use for notifying the sleep status of IoT node to client in the system.

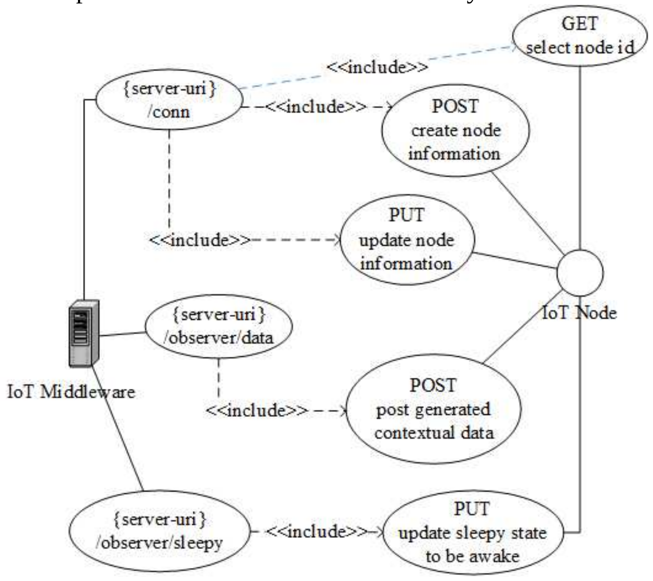


Fig. 6 Use case for CoAP resources of IoT middleware

Figure 7 is the database ER- diagram for database in the IoT middleware for saving related information of IoT nodes. In this ER-diagram, there are table t_node, table t_unit and table t_data.

Table t_node is use for saving IoT nodes information. The IoT node information involves node’s basic information, the IoT middleware URI, and node’s status information. “id” is the ID of the registered IoT node. “version” is used for identify the profile of the IoT node. “uri” is the node’s URI and “mw_uri” is the IoT middleware’s URI. “sleep_status”, “sleep_duration”, “notity_enable”, “notify_interval”, ”conn - ection_status” and “sleep_time” are used for sleepy mechanism.

Unit information are stored in Table "t_unit" which includes the following columns: ID, resource type, interface, and status information of unit. The unit can be sensors and actuators, such as temperature sensor, humidity sensor, fan, LED, etc. According to the IETF CoRE WG and other (OCF, oneM2M, etc) organization’s specifications, the resource type and interface are required for representation of sensors or actuators. The unit information is not much important for the sleepy mechanism. But, as a part of the IoT environment, the representation of those equipments are necessary. Therefore, the table includes status of the unit and data collection interval information.

Table t_data includes contextual data and the data inserted time for recording collection of environment data by units of the IoT node. These data from the IoT node, which formatted in JSON type and parsed in the IoT middleware and saving to the database. The real time data is saved in the IoT node by collection of units such as sensors. The IoT middleware request to the IoT node to get the data discontinuously. And we design the resource interface of the IoT node to support respond period data. The IoT node saves the real time environment data to the repository of the IoT node, and the IoT middleware can request the period data from IoT node. In this case, the inserted time is same for all the data is inserted which from the request.

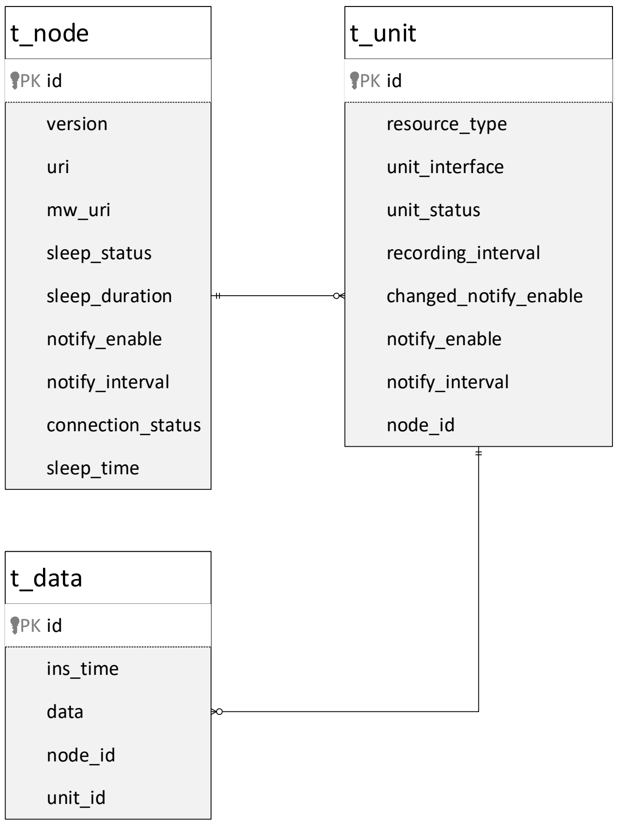
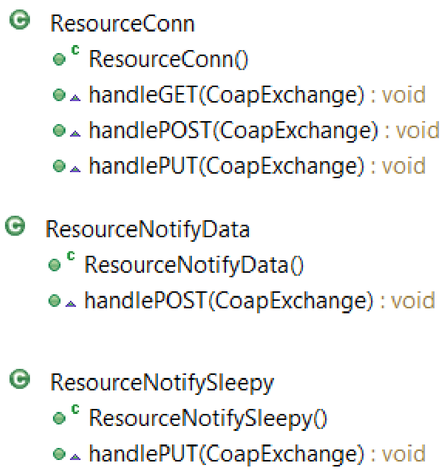


Fig. 7 ER-diagram of IoT middleware database

For implementing the IoT middleware, the CoAP resources should be implemented using Cf CoAP framework. These resource classes extend CoapResouce which is a basic implementation of a resource. Figure 8 shows the resources which extends class CoapResouce to implement the CoAP resource in the IoT middleware. Instances of type or subtype of CoapResource can be built up to a tree very easily. CoapResource uses four distinct methods to handle requests, that includes handleGET(), handlePOST(), handlePUT() and handleDELETE(). Each method has a default implementation that responds with a 4.05, that means the method is not allowed. Each method exists twice but with a different parameter, which are handleGET(Exchange) handleGET(CoAPExchange) for instance. The class is used internally in Cf to keep the state of an exchange of CoAP messages.



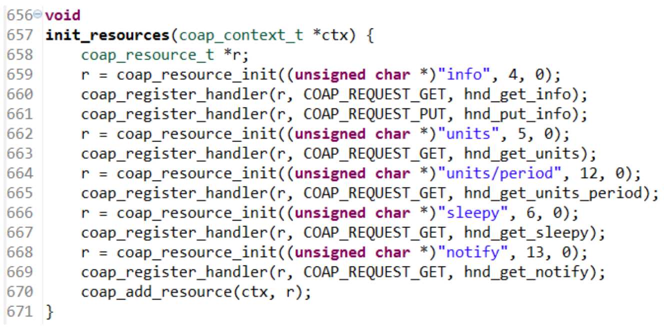
```
ResourceConn
    ResourceConn()
    handleGET(CoapExchange) : void
    handlePOST(CoapExchange) : void
    handlePUT(CoapExchange) : void

ResourceNotifyData
    ResourceNotifyData()
    handlePOST(CoapExchange) : void

ResourceNotifySleepy
    ResourceNotifySleepy()
    handlePUT(CoapExchange) : void
```

Fig. 8 CoAP resources of IoT middleware

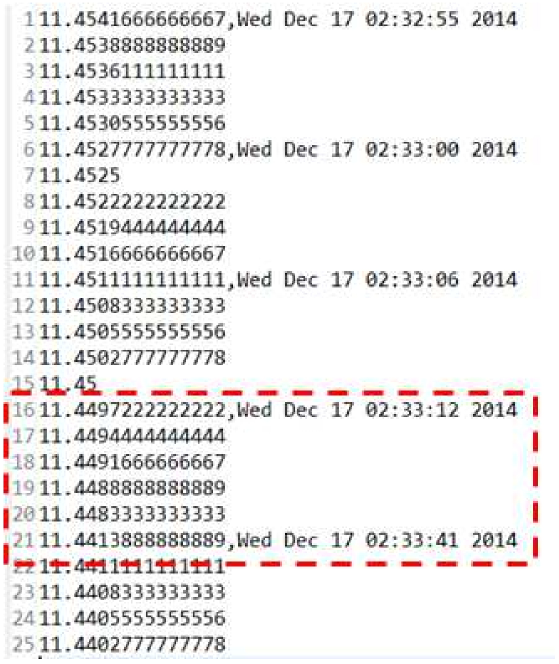
Figure 9 shows CoAP resource initialization functions of the IoT Node. The function coap_resource_init() is used for initializing CoAP resources which is supported by the library libcoap. The function coap_register_handler() is used for registering the CoAP resources to the CoAP server for providing CoAP services. The functions hnd_get_info, hnd_put_info, hnd_get_units, hnd_get_units_period, hnd_get_sleepy, and hnd_get_notify are handlers for the CoAP resources.



```c
void
init_resources(coap_context_t *ctx) {
    coap_resource_t *r;
    r = coap_resource_init((unsigned char *)"info", 4, 0);
    coap_register_handler(r, COAP_REQUEST_GET, hnd_get_info);
    coap_register_handler(r, COAP_REQUEST_PUT, hnd_put_info);
    r = coap_resource_init((unsigned char *)"units", 5, 0);
    coap_register_handler(r, COAP_REQUEST_GET, hnd_get_units);
    r = coap_resource_init((unsigned char *)"units/period", 12, 0);
    coap_register_handler(r, COAP_REQUEST_GET, hnd_get_units_period);
    r = coap_resource_init((unsigned char *)"sleepy", 6, 0);
    coap_register_handler(r, COAP_REQUEST_GET, hnd_get_sleepy);
    r = coap_resource_init((unsigned char *)"notify", 13, 0);
    coap_register_handler(r, COAP_REQUEST_GET, hnd_get_notify);
    coap_add_resource(ctx, r);
}
```

Fig. 9 Components of IoT middleware and IoT node

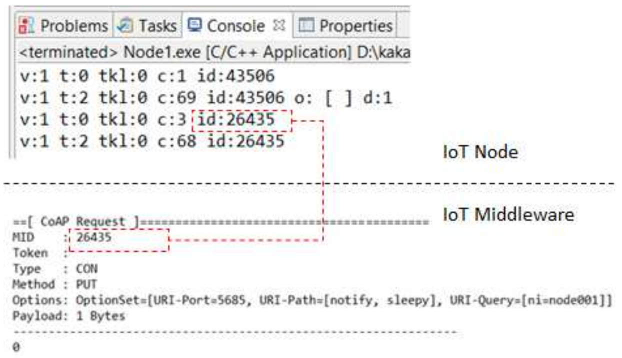
Figure 10 shows the sensing data record of IoT node for presenting the activity of IoT node in the sleepy mode.Once the IoT node receives the sleep command from the IoT middleware, the IoT node goes into the sleep mode. In other case the IoT node can go into the sleep mode based on the rule that is applied in the IoT node. In this case that is shown in the figure, the IoT node have received the request from the IoT middleware to go into the sleep mode for 15 seconds in sleep duration. In this period, the IoT node stops the communication and sensing or actuating functions. Therefore, record shows from “02:33:12” to “02:33:41” to sensing data is recorded.



```
 1 11.4541666666667,Wed Dec 17 02:32:55 2014
 2 11.4538888888889
 3 11.4536111111111
 4 11.4533333333333
 5 11.4530555555556
 6 11.4527777777778,Wed Dec 17 02:33:00 2014
 7 11.4525
 8 11.4522222222222
 9 11.4519444444444
10 11.4516666666667
11 11.4511111111111,Wed Dec 17 02:33:06 2014
12 11.4508333333333
13 11.4505555555556
14 11.4502777777778
15 11.45
16 11.4497222222222,Wed Dec 17 02:33:12 2014
17 11.4494444444444
18 11.4491666666667
19 11.4488888888889
20 11.4483333333333
21 11.4413888888889,Wed Dec 17 02:33:41 2014
22 11.4411111111111
23 11.4408333333333
24 11.4405555555556
25 11.4402777777778
```

Fig. 10 Sensing data record of IoT node in sleep mode

Once the client of service provide request to the IoT node in a sleep mode, the client cannot get any result from the IoT node. In this case, the client shall subscribe the IoT node through the IoT middleware. The client needs to request the IoT middleware the subscription process with the IoT node. For the process, the IoT middleware stores the information of request. After the IoT node notifies the wake-up status to the IoT middleware, the IoT middleware shall request to the IoT node for handling the subscription of client using the stored request information.



```
<terminated> Node1.exe [C/C++ Application] D:\kaka
v:1 t:0 tkl:0 c:1 id:43506
v:1 t:2 tkl:0 c:69 id:43506 o: [ ] d:1
v:1 t:0 tkl:0 c:3 id:26435
v:1 t:2 tkl:0 c:68 id:26435
```

```
==[ CoAP Request ]============================================
MID     : 26435
Token   :
Type    : CON
Method  : PUT
Options: OptionSet=[URI-Port=5685, URI-Path=[notify, sleepy], URI-Query=[ni=node001]]
Payload: 1 Bytes
---------------------------------------------------------------
0
```

Fig. 11 Wake-up message mapping of IoT middleware and IoT node

Figure 11 shows the CoAP message for notifying the sleep status to the IoT middleware. The CoAP message is sent by the IoT node to the IoT middleware for updating the sleep information in the IoT middleware. The log of CoAP message in the IoT node that includes the CoAP message ID for 26435, and the log of CoAP message in the IoT middleware that includes same CoAP message ID. The log of IoT middleware shows the resource URI is “notify/sleepy”, query is ni with the value node001. The message payload is 0, which means the IoT node sleep mode is wake up status.

IV. Conclusions

The IETF CoAP is a transmission protocol in the application layer that is used for the communications between devices in constrained environments. The IoT device equips with sensors and actuators to provide services ubiquitously to the users. Therefore, the device requires the small size with limited computing parts and power supply. With the sleepy mode, the IoT device can reduce the power resource more efficiently. We have proposed the sleepy scheme using the subscribe/publish communication architecture of MQ broker for the IoT device in the CoAP network. The subscribe/publish enables the client to request the IoT device in a sleep status through the IoT middleware. The IoT middleware includes the functionalities of RD and MQ broker to support the sleepy scheme through subscribe-publish communications to the client. Through the RD, the IoT middleware provide services of registering the IoT device information, and looking-up by the client. The information of sleepy mode also available in the IoT middleware for being retrieved by the client. From the IoT middleware, the client can subscribes the data that provided by the IoT device through the IoT service. Once the IoT device goes into the wake-up status from the sleep status, the IoT middleware gets the data and publishes to the client. Through the sleepy scheme in the CoAP network, we can apply more efficient energy consumption on IoT devices for having long life time in the constrained environment.

Acknowledgment

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2017-2016-0-00313) supervised by the IITP(Institute for Information & communications Technology Promotion), and this research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (2015R1D1A1A01060493), Any correspondence related to this paper should be addressed to DoHyeun Kim.

References

[1] Evans, Dave. "The internet of things: How the next evolution of the internet is changing everything." CISCO white paper1.2011 (2011): 1-11.
[2] Atzori, Luigi, Antonio Iera, and Giacomo Morabito. "The internet of things: A survey." Computer networks 54.15 (2010): 2787-2805.
[3] Jin, Wenquan, and Do Hyeun Kim. "Design and Implementation of e-Health System Based on Semantic Sensor Network Using IETF YANG." Sensors 18.2 (2018): 629.
[4] Series, M. "IMT Vision–Framework and overall objectives of the future development of IMT for 2020 and beyond." (2015).
[5] Jazayeri, Mohammad Ali, Steve HL Liang, and Chih-Yuan Huang. "Implementation and evaluation of four interoperable open standards for the internet of things." Sensors 15.9 (2015): 24343-24373.
[6] Kafle, Ved P., et al. "Scalable Directory Service for IoT Applications." IEEE Communications Standards Magazine 1.3 (2017): 58-65.
[7] Edwards, W. Keith. "Discovery systems in ubiquitous computing." IEEE Pervasive Computing 5.2 (2006): 70-77.
[8] Meshkova, Elena, et al. "A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks." Computer networks 52.11 (2008): 2097-2128.
[9] Bormann, Carsten, Angelo P. Castellani, and Zach Shelby. "Coap: An application protocol for billions of tiny internet nodes." IEEE Internet Computing 16.2 (2012): 62-67.
[10] Shelby, Zach, Klaus Hartke, and Carsten Bormann. "The constrained application protocol (CoAP)." (2014).
[11] Levä, Tapio, Oleksiy Mazhelis, and Henna Suomi. "Comparing the cost-efficiency of CoAP and HTTP in Web of Things applications." Decision Support Systems 63 (2014): 23-38.
[12] A. Rahman, "Sleepy Devices: Do we need to Support them in CORE?", draft-rahman-core-sleepy-nodes-do-we-need-01, February 11, 2014.
[13] Jin, Wenquan, and DoHyeun Kim. "A Sleep-Awake Scheme Based on CoAP for Energy-Efficiency in Internet of Things." JOIV: International Journal on Informatics Visualization 1.4 (2017): 110-114.
[14] Rahman, A. Enhanced Sleepy Node Support for CoAP. Internet-Draft, draftrahman-core-sleepy-05, 2014.
[15] Jin, Wen-Quan, and Do-Hyeun Kim. "Implementation and Experiment of CoAP Protocol Based on IoT for Verification of Interoperability." The journal of the institute of internet, broadcasting and communication 14.4 (2014): 7-12.
[16] 10] Shelby, Zach, Carsten Bormann, and Srdjan Krco. "CoRE resource directory." (2013).
[17] Koster, M., A. Keranen, and J. Jimenez. Message Queueing in the Constrained Application Protocol (CoAP). Internet-Draft, draft-koster-core-coapmq-00, 2014.
[18] Zakaria, Mohamad Fauzi, Joo Chin Shing, and Mohd Razali Md Tomari. "Implementation of Robot Operating System in Beaglebone Black based Mobile Robot for Obstacle Avoidance Application." International Journal on Advanced Science, Engineering and Information Technology 7.6 (2017): 2213-2219.
[19] Californium, http://people.inf.ethz.ch/mkovatsc/californium.php, 3.13.2018.
[20] Libcoap, http://libcoap.sourceforge.net, 3.13.2018.
[21] cJSON, https://sourceforge.net/projects/cjson, 3.13.2018.