symmetry

MDPI

Article

Data Aggregation Gateway Framework for CoAP Group Communications

Minki Cha [1], Jung-Hyok Kwon [1], SungJin Kim [2], Taeshik Shon [2] and Eui-Jik Kim [1,*]

[1] Department of Convergence Software, Hallym University, Chuncheon-si 24252, Korea; dxsaq12@hallym.ac.kr (M.C.); jhkwon@hallym.ac.kr (J.-H.K.)
[2] Department of Computer Engineering, Ajou University, Suwon-si 16502, Korea; ksjskyblue@ajou.ac.kr (S.J.K.); tsshon@ajou.ac.kr (T.S.)
* Correspondence: ejkim32@hallym.ac.kr; Tel.: +82-33-248-2333; Fax: +82-33-242-2524

Academic Editors: Ka Lok Man, Yo-Sub Han and Hai-Ning Liang
Received: 7 August 2016; Accepted: 18 November 2016; Published: 24 November 2016

Abstract: In this paper, a data aggregation gateway framework (DA-GW) for constrained application protocol (CoAP) group communications is proposed. The DA-GW framework is designed to improve the throughput performance and energy efficiency of group communication to monitor and control multiple sensor devices collectively with a single user terminal. The DA-GW consists of four function blocks—the message analyzer, group manager, message scheduler and data handler—and three informative databases—the client database, resource database and information database. The DA-GW performs group management and group communication through each functional block and stores resources in the informative databases. The DA-GW employs international standard-based data structures and provides the interoperability of heterogeneous devices used in various applications. The DA-GW is implemented using a Java-based open source framework called jCoAP to evaluate the functions and performance of the DA-GW. The experiment results showed that the DA-GW framework revealed better performance than existing group communication methods in terms of throughput and energy consumption.

Keywords: CoAP group communication; data aggregation; gateway framework; Internet of Things; jCoAP; oneM2M

1. Introduction

Recently, Internet of Things (IoT) technology has undergone rapid advancement and enabled a large number of services, such as smart manufacturing, smart agriculture, healthcare and connected cars [1–3]. In such IoT services, devices obtain data from physical sensors and conduct the operations requested by the user through the actuators, then report the obtained data and the operating results to the user. These devices are generally resource-constrained in central processing unit (CPU), memory and power consumption, and the use of a light-weight web protocol is thus strongly recommended to make their data resources accessible to the Internet [4].

The constrained application protocol (CoAP) is a typical light-weight web transfer protocol for providing reliable communication in a resource-constrained environment. It is specified in Request for Comments (RFC) 7252 published by the Internet Engineering Task Force (IETF) Constrained RESTful Environments (CoRE) Working Group and has been standardized [5]. Since the CoAP runs over the User Datagram Protocol (UDP), it has a smaller control overhead than the Hypertext Transfer Protocol (HTTP), which is a Transmission Control Protocol (TCP)-based web transfer protocol. It can also reduce network overhead because of the small size of the message header (i.e., four bytes) [6]. Furthermore, since it follows the RESTful structure, it can be easily converted and interoperated with the existing HTTP web protocol, thereby providing high interoperability [7].

In most IoT services, the function of group communication is highly important for enabling a single user terminal (i.e., client) to monitor and control multiple sensor devices (i.e., servers) collectively. To achieve this, the IETF CoRE Working Group published RFC 7390 to standardize multicast-based CoAP group communication [8]. In the standard, the client manages the data resources of servers in the designated group as it transfers non-reliable multicast messages to the designated group. However, the CoAP group communication defined in RFC 7390 cannot guarantee reliable transmission and has the drawback of difficulties in implementing multicast transfer over the resource-constrained environment.

A number of studies has been conducted to improve CoAP group communication. Konieczek et al. [9] and Law et al. [10] proposed a unicast-based solution. In the solution, the client sends a request directly to each server that belongs to the group and receives a response individually from each server. In addition, once the client receives a response from all servers in the group, it determines that communication with the group has been completed. In this way, the client can request re-transmission if errors are detected, as it can determine immediately whether it has received all responses from each server, thereby ensuring reliable communication. However, the client needs to transfer a large number of request messages and receives responses from each server individually, which can cause network congestion, as well as high energy consumption in the client. Moreover, a long delay due to repetitive request-response operations cannot be avoided, which cannot guarantee the timeliness of IoT services. To solve this problem, Ishaq et al. [11] proposed a gateway-centric unicast solution. In this solution, additional gateways (GWs) are placed between the client and servers. The client sends a request message that contains the information of the servers to the GW only once. The GW sends a request directly to each server listed in the request message via unicast and forwards the response to the client as soon as it receives a response. In this way, the number of request transmissions sent by the client can be reduced, thereby mitigating network congestion between the client and GW and increasing the energy efficiency of the client. However, it cannot reduce the number of request transmissions between the GW and servers, and the number of responses received by client and GW is not taken into consideration. As a result, the above method cannot solve the network congestion problem completely, and energy consumption is not reduced due to response-receiving. Jung et al. [12] and Ishaq et al. [13] commonly proposed a gateway-centric multicast solution. In the former solution [12], the GW sends a client request to all servers included in the group via multicast transfer once. It uses the "non-confirmable message" of CoAP as a client request, which does not require an acknowledgement from the servers. Thus, the client should send the additional "confirmable message" to servers to retrieve their resource, which can act as the network overhead. The latter [13] introduced a multicast solution for the device discovery procedure, for which the GW sends the "confirmable message" as a discovery request to the servers via a multicast transfer, and then, each server responds to the GW with an acknowledgement using the unicast. This solution cannot solve the network congestion problem between the client and GW, since it focuses on only the message exchange between the GW and servers for the device discovery procedure.

In the present study, the data aggregation gateway (DA-GW) framework for CoAP group communications is proposed to solve the aforementioned CoAP group communication issue. The DA-GW framework consists of essential functional blocks and an informative database (DB) for the client to manage multiple groups and various server devices efficiently that are included in each group through multicast. The DA-GW includes the following four functional blocks: (1) message analyzer; (2) group manager; (3) message scheduler and (4) data handler. The message analyzer analyzes request/response messages and determines what actions are taken by the DA-GW, and the group manager manages device discovery and the join/leave operations of servers (i.e., group or non-group members). The message scheduler determines the transmission time of request/response messages, and the data handler performs computation for data aggregation and data processing. In addition, the DA-GW includes the following three informative DBs: (1) client DB; (2) resource DB and (3) information DB. The client DB stores the information of user profiles, and the resource DB stores a list of all servers and their resources. Finally, the information DB stores processed data, such as minimum, maximum and mean values.

The DA-GW employs the oneM2M standard-based data structure to provide interoperability for heterogeneous devices that use a variety of applications. The DA-GW stores all of the generated data in the format of resources and distinguishes each resource by the uniform resource identifier (URI). DA-GW performs group management using the <groupManagement> resource and stores data by group for CoAP group communication. Furthermore, the DA-GW performs user authentication of the client through the <userAuthentification> resource and modifies the status of servers through application resources, such as <light>, <door> or <temperature>, or receives status values.

To verify the performance of the DA-GW framework, a prototype was implemented using jCoAP, a Java-based open source, and performances were measured in terms of throughput and energy consumption and compared with existing solutions. The experiment results showed that the DA-GW revealed better throughput performance than that of the unicast solution by 73.71% and that of the multicast solution by 19.47% when the timeout was set to 60 ms, the number of groups was two and the number of members in each group was set to nine. Under the same experimental conditions, the energy consumption was decreased by 13.37% compared with that of the unicast solution and by 2.98% compared with that of the existing multicast solution.

The present paper is organized as follows. In Section 2, the system architecture and the design of the DA-GW are explained in detail. In Section 3, the implementation environment of the DA-GW is presented with functions and performance evaluation results. We conclude this paper in Section 4.

2. Data Aggregation Gateway Framework

2.1. System Architecture

Figure 1 illustrates the high-level system architecture of the CoAP group communication described in our work, which consists of three components: the client, the DA-GW and the servers. In the figure, the servers (i.e., the sensor and actuator devices) are connected to the client (i.e., the user terminal) through a DA-GW.

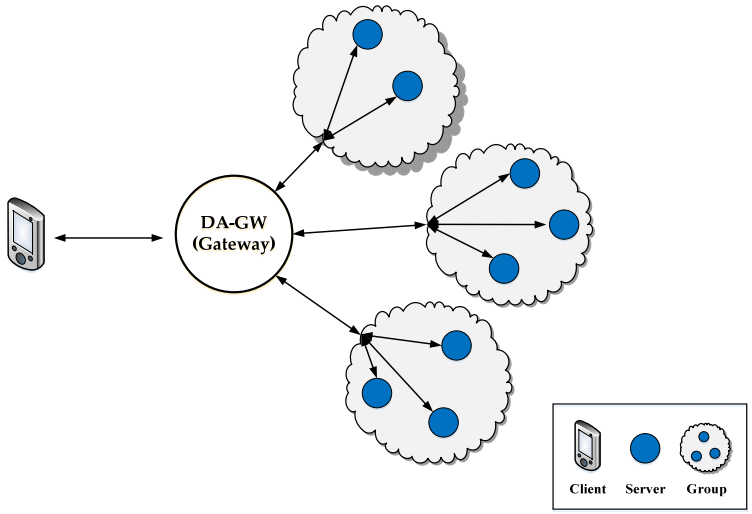


Figure 1. High-level system architecture.

One or more servers with a similar task within a local network can compose a group. When the DA-GW receives the request message from a client, it forwards it to the servers within the designated group using multicast transmission, and then, each server responds with the response message that includes the requested sensing data using unicast transmission. In this paper, the DA-GW plays the role of an aggregation point. In other words, the DA-GW checks and collects data from each server within the group. Furthermore, it checks the group address contained in the header of each response

message and aggregates response messages that contain the same group address to a single message (i.e., aggregation message). The DA-GW performs aggregation after waiting for a response message within the pre-set timeout period. Through this process, network traffic can be reduced.

2.2. Design of DA-GW

Figure 2 shows the system structure of the DA-GW. The DA-GW consists of four functional blocks and three informative DBs.

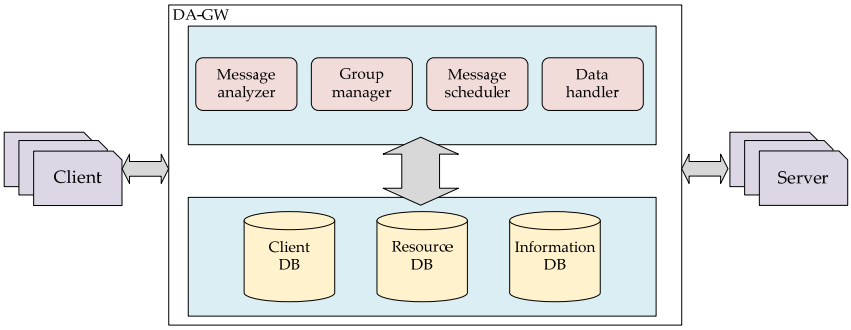


Figure 2. System structure of the DA-GW.

- Message analyzer: The message analyzer checks the type of received request/response message and determines the operation undertaken by the DA-GW. The message analyzer classifies a request message into three types: (1) resource discovery request; (2) join/leave group request and (3) group communication request. When the DA-GW receives a response message from the servers, the message analyzer determines whether the received response message has an error and whether aggregation is performed by checking the group URI.
- Group manager: The group manager performs device discovery and join/leave group operations. Once a change in a group is determined through the message analyzer, the DA-GW gives a request for an operation to a corresponding server. For example, once the client requests the addition of a new member to an existing group, the DA-GW forwards the join group request received from the client to a corresponding server. Once a member is joined to the group normally, the DA-GW gives the response of the changed status of a group to the client.
- Message scheduler: The message scheduler determines the transfer time of the request/response message between the client and server. The DA-GW considers network traffic to adjust the transfer time of the request/response message between the client and server when its buffer overflow is imminent. To this end, the DA-GW configures the threshold of its own buffer queue, which is a reference value for determining whether a buffer overflow event is imminent. Note that, in our work, we use 90% of the buffer size as a queue threshold. When the queue status of the DA-GW reaches the pre-defined queue threshold, it immediately sends the aggregated response message to the client without waiting for further responses, even if the timeout period is not expired yet.
- Data handler: The data handler performs computation for the aggregation or processing of the response messages received from the servers. The data handler aggregates the response messages that contain the same group address within the pre-set timeout and sends a response to the client. If the payload size of the aggregated response messages is larger than the maximum payload size, it separately transmits a response having a maximum payload and an additional response that contains the rest of the payload. Furthermore, it processes the received data into special information, such as minimum, maximum or mean values.
- Client DB: The client DB stores the information of user profiles. Once user authentication is received from the client, the DA-GW compares the user profile information with that in the client DB to verify the user. Through this process, the DA-GW supports multi-clients.

- Resource DB: The resource DB stores a list of group members and non-group members, as well as the resources of each member. The DA-GW provides the resource information stored in the resource DB to the client for group management. The DA-GW updates the resource DB whenever changes in resources occur.
- Information DB: The information DB is an optional DB that stores processed data, such as minimum, maximum and mean values. Once the client requests processed data, the DA-GW gives a response regarding the processed data stored in the information DB immediately. The DA-GW creates processed data via requests from the client and sets the update frequency.

2.3. Data Structure

The data structure in the DA-GW follows the resource structure defined in oneM2M TS-0001 [14]. Through the data structure, the information used in the DA-GW and servers can be explained. Information about user authentication is expressed via the string type. Information about the group server and non-group server is expressed via the string type. Furthermore, group management information for group join/leave functions and group communication is also expressed via the string type. The sensor value is the real type, and the actuator status is the string type. The period of observation is expressed via the integer type, and the observation status is expressed as integer Type 0 or 1.

Figure 3 shows the resource structure of user authentication used in the DA-GW. The accessedClientIDList attribute shows a list of client IDs connected to the DA-GW. The registeredClientIDList attribute shows the client IDs registered to the DA-GW. The userAuthenticationState attribute represents a value showing whether log-in has been successful. It checks whether the client is connected to the DA-GW using the accessedClientIDList value and registeredClientIDList value and modifies the value for log-in success after checking whether the ID is registered to the DA-GW. The <subscriptions> sub-resource of a specific resource creates the <subscription> resource and gives a response when it receives the subscription request of the client. Once an attribute value of the client subscription is changed, the corresponding client is notified of the change. All <subscriptions> sub-resources are operated similarly. Once the accessedClientIDList, registeredClientIDList and userAuthenticationState values are updated, the client that wants a subscription is notified of the update.

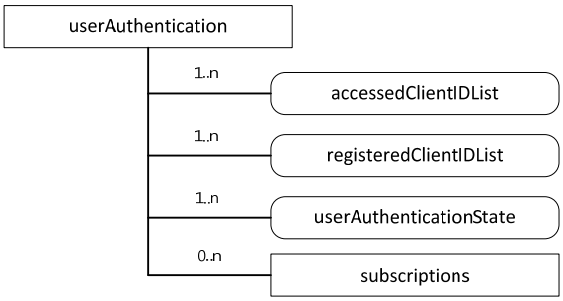


Figure 3. Data structure for authentication.

Figure 4 shows the data structure for group management. The nonGroupDevicesList attribute shows information about devices that do not belong to a specific group. The groupDevicesList attribute shows information about devices that belong to a specific group. The <groupAction> resource is a sub-resource of the <groupManagement> resource. The groupActionState attribute shows whether join or leave has been successful. It generates an event after receiving a request of join or leave to/from a specific group and modifies the groupActionState attribute to show whether that event has been successful. The <groupCommunication> resource, which is a sub-resource of the <groupManagement> resource, contains the multiple attributes that represent the values obtained

through group communication with sensor and actuator groups. In the figure, temperatureGroupState, lightGroupState and doorGroupState show the attributes for temperature sensor group, light actuator group and door actuator group as the references for guidance, respectively. Note that this configuration of the data structure is not restricted to the specific sensors or actuators.

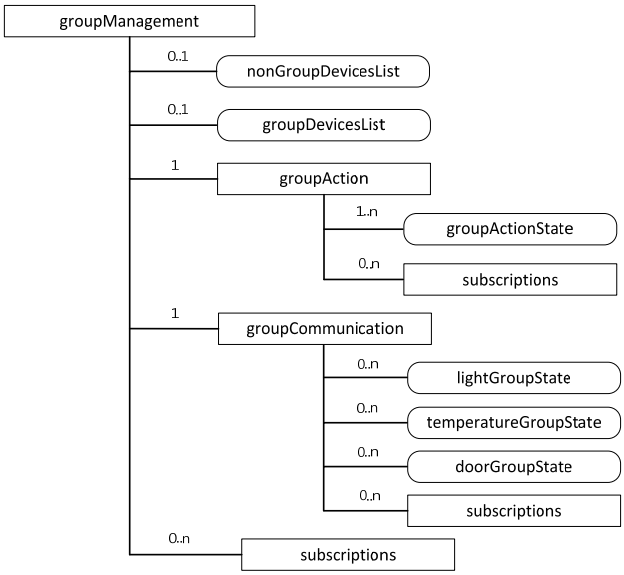


Figure 4. Data structure for group management.

Figure 5 shows the data structure for the server included in the light group. The lightActionState attribute shows the current state of the light. The <lightAction> resource is a sub-resource of the <light> resource. The client changes the state of the light to on or off and checks whether the light state has been changed successfully through the changeState attribute.

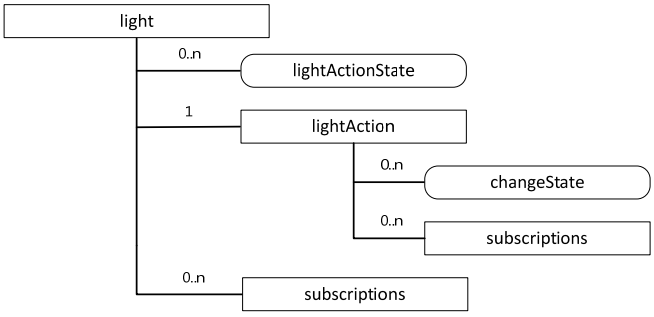


Figure 5. Data structure of the light application.

Figure 6 shows the data structure for the server included in the door group. The doorActionState attribute shows the current open/close state of the door. The <doorAction> resource is a sub-resource of the <door> resource. It performs a request that changes a door state from the client and can determine whether the operation has been successfully performed through changes in the value of the changeState attribute.

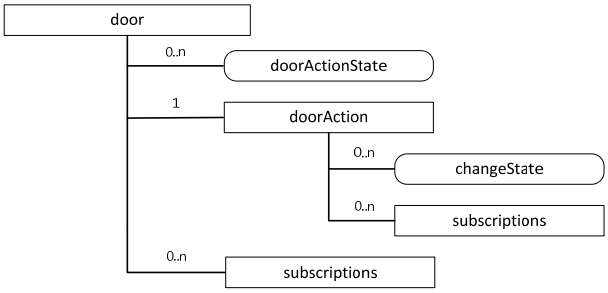


Figure 6. Data structure of the door application.

Figure 7 shows the data structure for the server included in the temperature group. The temperatureState attribute shows the temperature value. The notificationPeriod attribute shows the minimum time between notifications. The temperatureActionState attribute shows the state of the notification. The <temperatureAction> resource is a sub-resource of the <temperature> resource. The notificationEnable attribute shows that a periodic notification is being used. The notificationDisable attribute shows that a periodic notification is not being used.

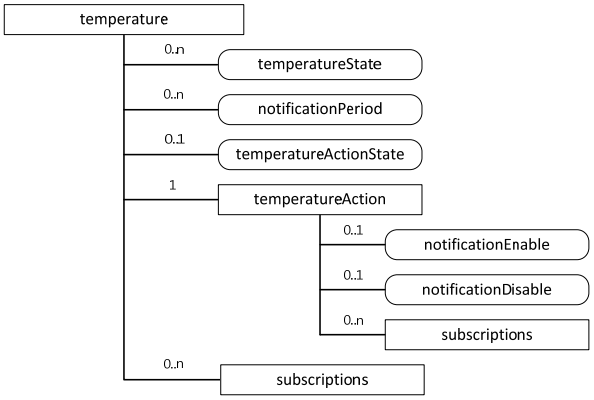


Figure 7. Data structure of the temperature application.

3. Implementation and Performance Evaluation

In this section, the functions and performances of the DA-GW are evaluated through implementation with extensive experiments. To evaluate the DA-GW, the implementation was conducted using jCoAP [15,16], which is a Java-based open source software, and the experimental results were compared with existing solutions to verify the performance of the proposed DA-GW. Next, the implementation is explained in detail, and the functions and performance evaluation results of the DA-GW are presented in the subsection.

3.1. Implementation

The CoAP group communication system was implemented, as shown in Figure 8. The DA-GW was placed between the client and server, and communication between them can be achieved through the DA-GW. Each device communicated using the CoAP over the Institute of Electrical and Electronics Engineers (IEEE) 802.11n wireless network interface [17,18]. The client transfers a request message of the JavaScript Object Notation (JSON) type that has a 52-byte payload to the DA-GW [19]. Once the DA-GW receives a request message from the client, the URI, URI query and payload are verified, and request messages are transferred to the servers using multicast. Each server that receives the request creates a response message of the JSON type that has a 20-byte payload and transfers the message to the DA-GW using unicast. The DA-GW aggregates response messages received from each server and transfers a single aggregated response message to the client.

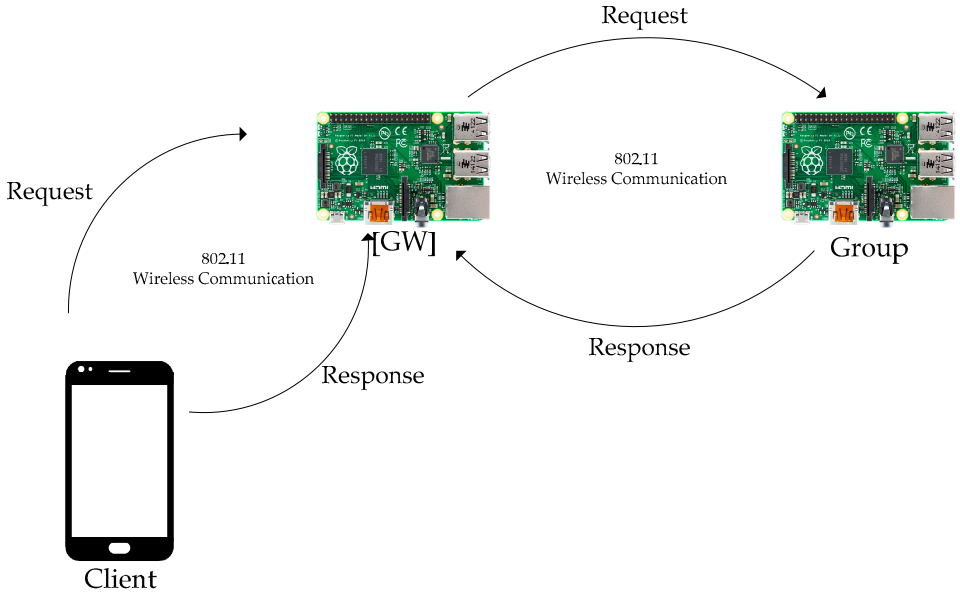


Figure 8. Implementation of the CoAP group communication system.

The DA-GW plays an important role in the CoAP group communication system. To implement the DA-GW, the Raspberry pi2 hardware platform (Raspberry Pi Foundation, Caldecote, Cambridgeshire, UK), which was run with a Linux-based Raspbian operating system (Raspberry Pi Foundation, Caldecote, Cambridgeshire, UK), was used. Raspberry pi2 supports a 900-MHz quad-core ARM Cortex-A7 CPU and 1 GB RAM [20]. To provide wireless connectivity, a wireless adapter that supported IEEE 802.11n was employed. The functional blocks that were responsible for the critical functions of the DA-GW, such as message analysis, transmission scheduling, group management and data aggregation, were implemented using the jCoAP library. To access the informative DB in the DA-GW, Java Database Connectivity 4.2 (JDBC 4.2) (Oracle Corporation, Santa Clara, CA, USA), which is an application programming interface (API) used to support DB access in Java, was used. To verify the functions of the DA-GW, the user interface (UI) was developed using the Swing toolkit provided by Java.

The server was also implemented using Raspberry pi2 and the jCoAP library. However, a sensor or actuator was attached to each server in contrast with the DA-GW, and the applications were developed differently according to the types of sensor and actuator applied to the server. More specifically, each server uses only one of the applications from the light application for LED actuator control, door application for servo motor control and temperature application for temperature sensor monitoring.

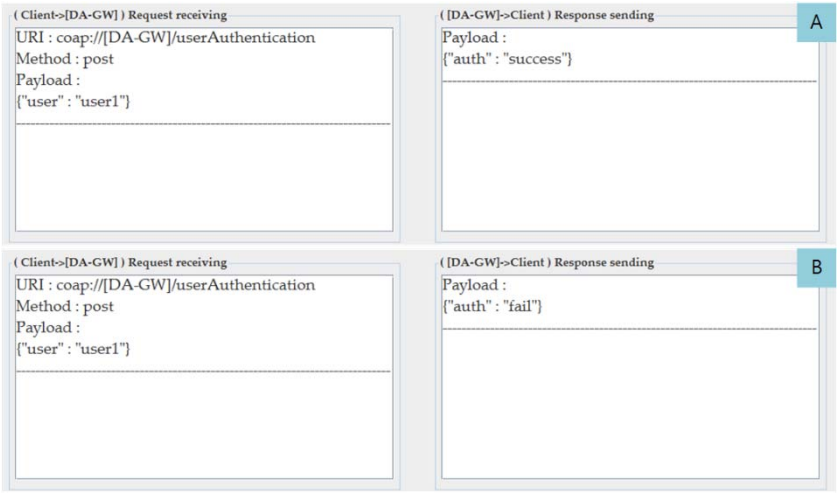
To implement the client, an Android Version 5.1.1 (i.e., Lollipop)-based tablet PC (i.e., Nexus 7) was used, and the client application was developed via the jCoAP library. The client employs the QUALCOMM (San Diego, CA, USA) WCN3660 wireless local area network (WLAN) interface, which supports Modulation and Coding Scheme 7 (MCS 7) in IEEE 802.11n.

3.2. Functional Evaluation

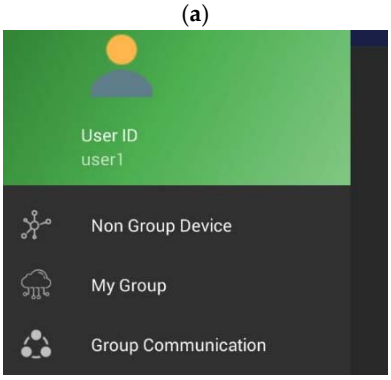
In this section, the main functions and operations of the DA-GW are evaluated. In particular, the functions of user authentication for multi-clients, group management and group communication in the DA-GW are checked through the UI.

Figure 9a shows the performance results of user authentication used in the DA-GW. The client requests authentication via the POST method with a JSON-type message containing the payload of the user ID by means of the "/userAuthentication" URI of the DA-GW. Then, the DA-GW compares the information in the payload in the request with that in the client DB. Upon successful authentication, "success" is contained in the payload of the JSON-type message, as shown in A, to give a response to the client. Otherwise, "fail" is contained in the payload, as shown in B, in response to the client. Upon

successful authentication, the client displays the UI, as shown in Figure 9b. Note that, in our work, we have mainly focused on the operation of data aggregation for CoAP group communications and its system prototyping considering the oneM2M standard and the jCoAP open source project for the purpose of proof of concept (PoC). Thus, regarding the user authentication, the DA-GW requires only the user ID for user authentication without any password or data encryption. However, in the real deployment of the DA-GW, this simple authentication is very vulnerable to malicious attacks, thus security and privacy should be essentially considered from the user service aspects.



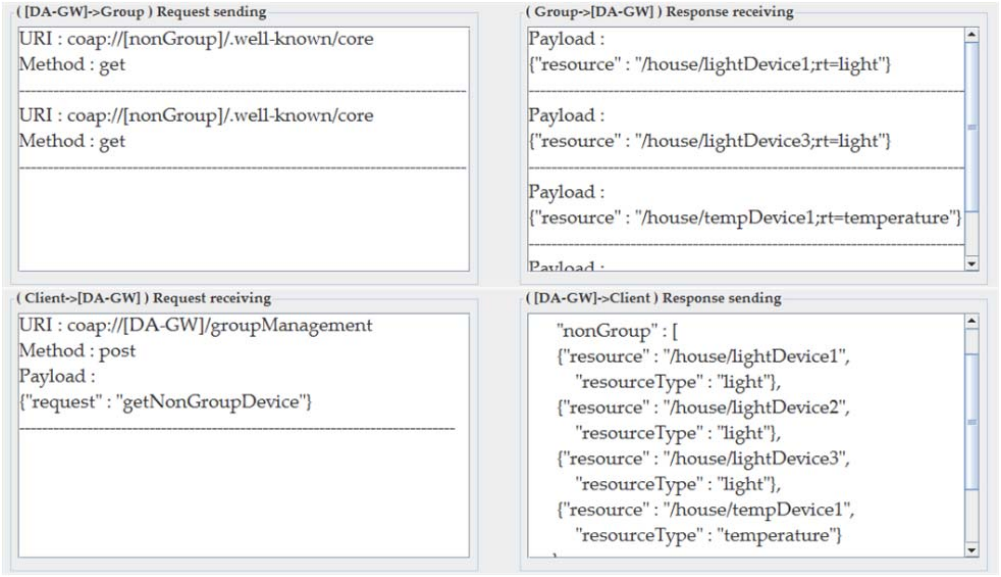
(**a**)



(**b**)

Figure 9. User authentication for multi-clients. (**a**) The result of user authentication in DA-GW (A: Success, B: Fail); (**b**) the client UI when the user is authenticated successfully.

The group management function includes device discovery and join/leave group functions. Figure 10a shows the performance results of device discovery used in the DA-GW. The DA-GW performs device discovery periodically to search a newly-added server in the service region. The DA-GW transfers a discovery request message via the "well-known/core" URI for device discovery. If servers that are not subscribed to the group receive the message, the discovery response message is transferred to the DA-GW. Once the DA-GW receives the response message, the corresponding server is stored in the resource DB as a non-group member. Once device discovery is performed successfully, the resource type of the non-group member can be seen in the client, as shown in Figure 10b.

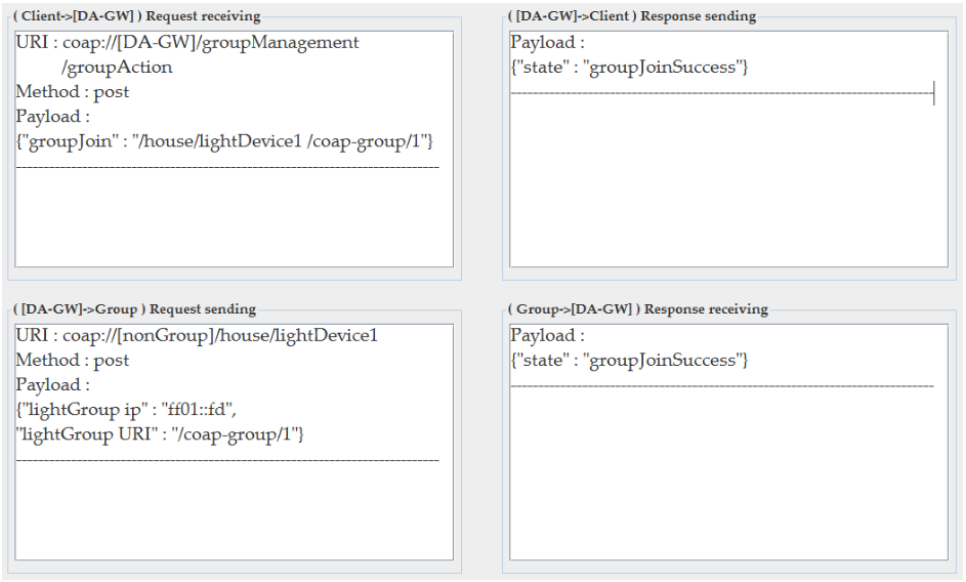


(**a**)

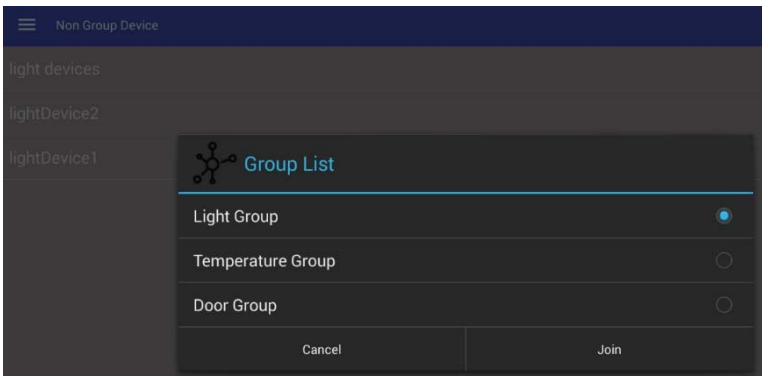
(**b**)

Figure 10. Device discovery. (**a**) The result of non-group member discovery in DA-GW; (**b**) the resource type of the non-group member.

Figure 11 shows the performance result of the join group. As shown in Figure 11b, once a user selects the resource type depicted in the client, a list of discovered non-group members is displayed. Once one of the non-group members in the list is selected by a user, the list of groups to which the selected non-group member can be registered is displayed. Then, a specific group in the group list is selected, and the join button is selected. The client transfers the URI of the non-group member selected via the "/groupManagement/groupAction" URI of the DA-GW and a join request message including the group URI through the POST method. Once the DA-GW receives the request, the group IP and group URI are transferred to the corresponding non-group member URI. After the corresponding non-group member is joined to the group selected by user, the non-group member gives a response about the group join state to the DA-GW. Then, the DA-GW that receives the response stores the group join information in the resource DB. The leave process is similar to the aforementioned join process. The client sends a leave request from the group to the "client/GM" URI of the DA-GW with group member information that needs to leave from the group using the POST method. Once the DA-GW receives the message, it gives a leave request to corresponding group members, and a group member that receives the message updates its state to non-group member and transfers the information to the DA-GW. The DA-GW that receives the response deletes the group members that have left from the resource DB.



(**a**)



(**b**)

Figure 11. Join group. (**a**) The result of the group join process in DA-GW; (**b**) the group management of the client.

Figure 12 shows the performance result of group communication. As shown in Figure 12b, when the client changes the state of a group member, it sends a request message to the DA-GW using the POST method. Once the DA-GW receives the request message, the request received from the client is transmitted to group members via multicast. Once the group members receive the request message, they perform the requested operation, and a response on the performance result is sent to the DA-GW via unicast individually. The DA-GW that receives responses from the group members performs data aggregation until either the response messages are received from all group members or timeout occurs. Then, the DA-GW transfers the aggregated response message to the client only once.

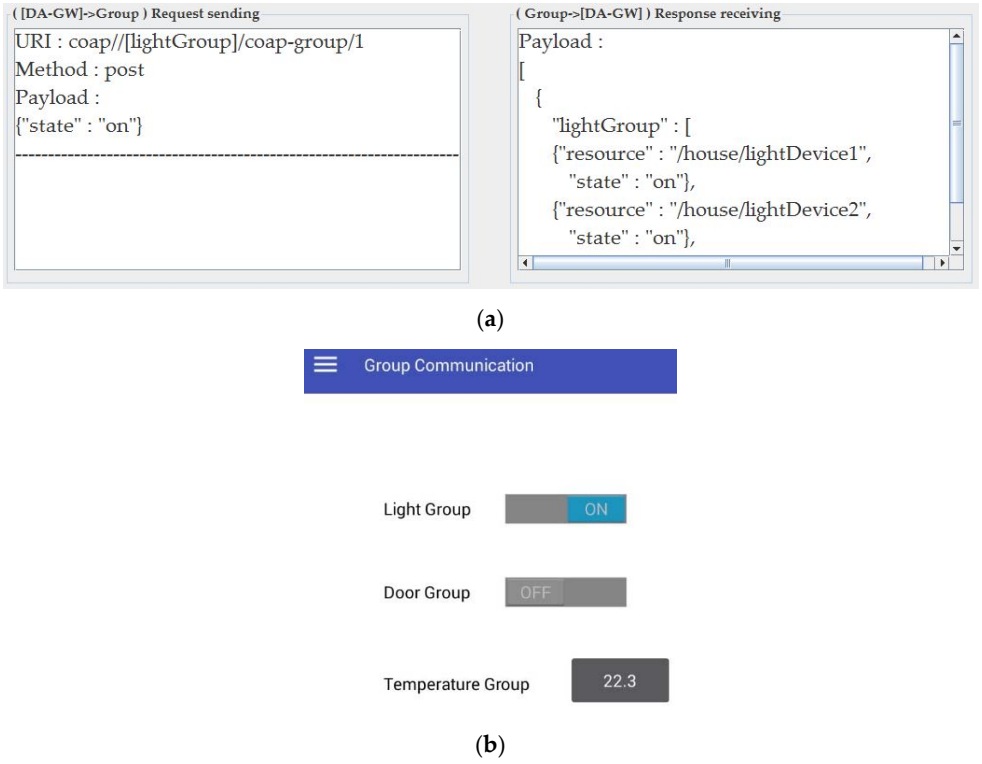


Figure 12. Group communication. (**a**) The result of group communication in DA-GW; (**b**) the group communication in the client.

3.3. Performance Evaluation

In this section, we evaluate the performance of the DA-GW by conducting the experiments for three approaches; "gateway-centric unicast", "gateway-centric multicast" and "DA-GW". As earlier mentioned in Section 1, in the gateway-centric unicast, the GW is placed between the client and servers. The client sends a request message to the GW only once, then the GW sends a request directly to each server via unicast and forwards the response to the client as soon as it receives a response. In the gateway-centric multicast approach, the GW sends a request from the client to all servers included in the group via multicast transfer once, but it separately forwards the response via unicast whenever receiving a response from each server. Lastly, in the DA-GW approach, the GW performs the aggregation for multiple response messages from the servers during the pre-set timeout period and then forwards an aggregated message to the client.

Figure 13 shows the delay time to receive all responses from all group members after the client sends the request and the energy consumption in the client due to the delay. The payload sizes of the request and response message are 52 and 20 bytes, respectively. The request message contains the information for the server (e.g., URI path, group communication IP, operation command), while the response message includes the measured value of the sensor or actuator. Note that the Tx energy consumption in the client (i.e., Nexus 7) is 138.125 mW, and the *Rx* energy consumption is 76.778 mW. Considering the effect of the group size, the number of group members started from two up to 20 incrementing by two, and the number of groups was set to one. Each experiment was conducted for 5 min and iterated 10 times.

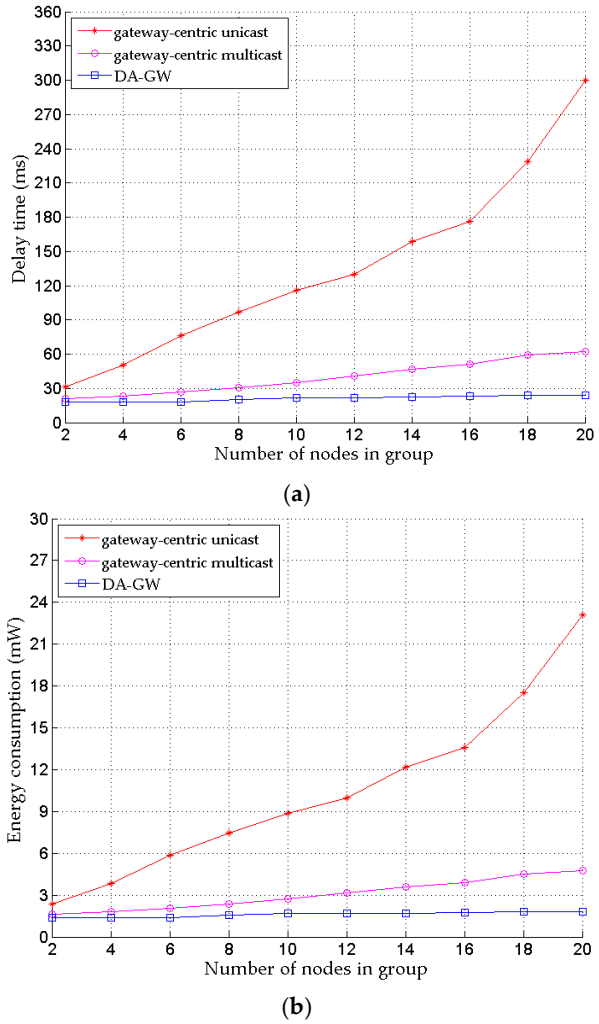


Figure 13. Delay time and energy consumption for group communication. (**a**) Delay time; (**b**) energy consumption.

As shown in Figure 13a, as the number of group members was increased, the delay time increased. However, group communication using the DA-GW had a shorter delay time than existing methods. The reason for this was that the number of request message transmissions was reduced by using multicast, and the number of receipts of response messages was also reduced by aggregation in the case of the DA-GW. For gateway-centric unicast, on the other hand, as the number of group members was increased, the number of request message transmissions and the number of receipts of response messages were proportional to the number of group members. Additionally, for gateway-centric multicast, the number of receipts of response messages increased by the same amount as the number of group members. As shown in Figure 13b, energy consumption in the client showed a similar trend to that of the changes in delay time according to the increase in the number of group members. This was because increases in transmissions and receipts meant more energy consumption proportionally. Group communication using the DA-GW had a shorter delay time than those of gateway-centric unicast and gateway-centric multicast by 84.53% and 46.86%, as well as less energy consumption by 84.54% and 46.85%, respectively.

Figure 14 shows the throughput performance of group communication using the DA-GW. Figure 14a shows throughput according to increases in the number of group members. In this experiment, the timeouts were set to 20 and 40 ms. The client was set to transfer a request message with the frequency of timeout. Group communication using the DA-GW increased the number of

received response messages as the number of group members was increased, increasing throughput. On the other hand, the existing method showed that throughput increased until the number of group members exceeded a certain number. After this, a certain throughput was revealed. This was because all response messages could not be received within the timeout. The throughput in the method using the DA-GW was better than that of gateway-centric unicast by 83.30% and that of gateway-centric multicast by 56.36% when the timeout was set to 20 ms.

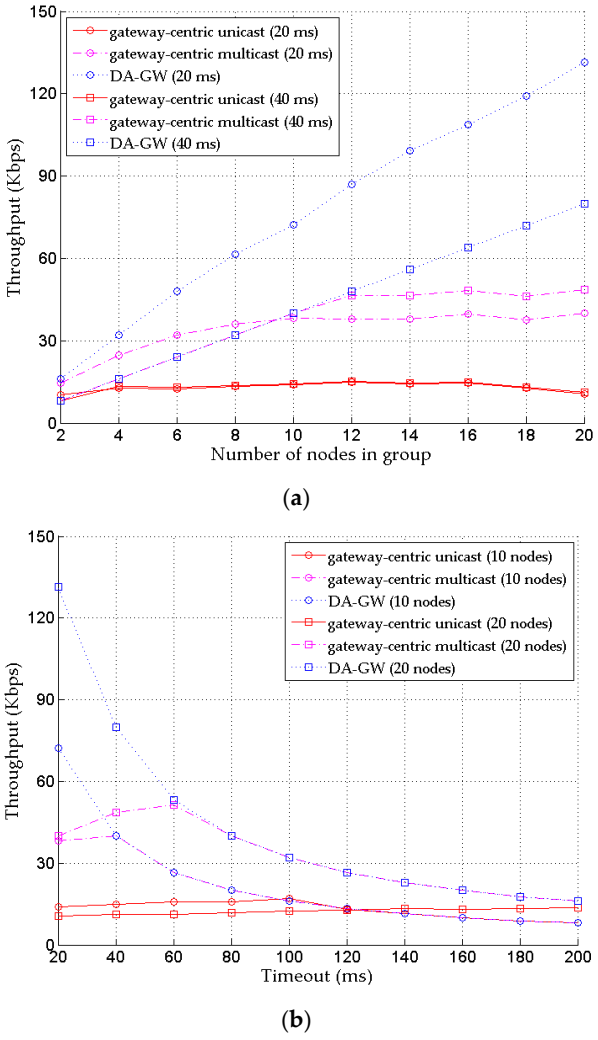
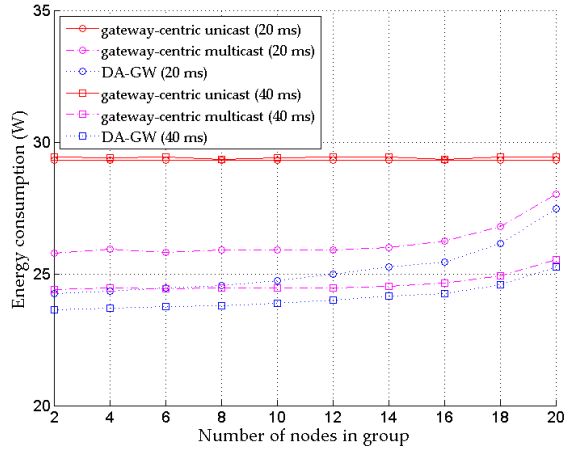


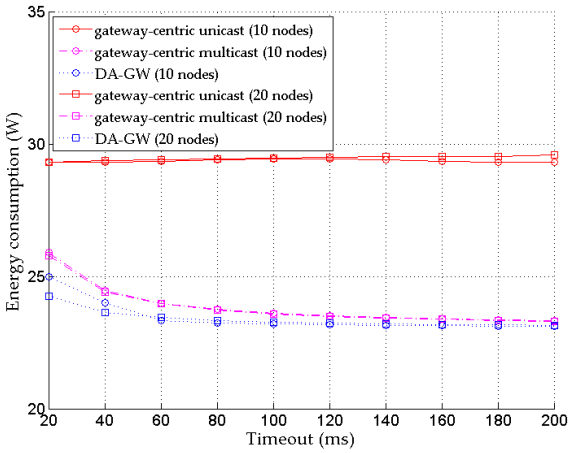
Figure 14. Throughput. (**a**) Throughput for a variation in the number of nodes in the group; (**b**) throughput for a variation in the timeout.

Figure 14b shows the throughput performance according to increases in timeout. In the experiment, the numbers of group members were set to 10 and 20, respectively. The experiment result showed that as the timeout increased, the throughput of the DA-GW decreased. This was because the number of response messages received by the client was reduced during the experiment. The proposed method showed higher throughput performance than existing methods when the timeout was short. This was because the method using the DA-GW received more response messages within the limited timeout. On the other hand, when the timeout was long, all three methods showed similar throughput performance because all three methods can receive all response messages. The throughput in the method using the DA-GW was better than that of gateway-centric unicast by 71.94% and that of gateway-centric multicast by 28.37% when the number of group members was 20.

Figure 15a shows the energy amount consumed by the client for 5 min according to changes in the number of group members. The experiment was conducted in the same environment to that shown in Figure 14a. When the DA-GW was used, the energy consumption increased as the number of group members was increased. This is because the energy consumed during the reception of messages increased as the number of received response messages increased. When the DA-GW was used, the client performed the least number of message transmissions and receptions, which was why the lowest energy was consumed compared with existing methods. On the other hand, when gateway-centric unicast was used, the energy consumption showed no significant change because the client showed little change in the number of transmissions and receptions regardless of the number of group members. For gateway-centric unicast, the energy consumption in the client was the largest because it had to send as many request messages as the number of group members. Figure 15b shows the energy consumption of the client according to an increase in timeout. The experiment was conducted in the same environment to that shown in Figure 14a. When the DA-GW was used, the energy consumption in the client continued to decrease as the timeout became longer, but it became nearly constant after the timeout passed 60 ms. The reason for this was because all response messages could be received if a timeout was set to 60 ms or longer, so there was no change in the number of receptions of response messages. For group communication using the DA-GW, the energy consumption in the client was approximately 18.03% less than that of gateway-centric unicast and 2.15% less than that of gateway-centric multicast.



(**a**)



(**b**)

Figure 15. Energy consumption. (**a**) Energy consumption for a variation in the number of nodes in the group; (**b**) energy consumption for a variation in the timeout.

Figure 16 shows the measurement results of (a) throughput and (b) energy consumption for 5 min of the client while increasing the number of groups from one to three. In this experiment, we considered the variation of the number of groups to investigate the effect of the number of groups. The number of overall servers was set to 18, and the timeout was set to 60 ms. The numbers of members in a group were set to 18, 9 and 6, when the number of groups was one, two and three, respectively. Each group sent request messages sequentially to transmit request messages to multiple groups. That is, as the number of groups was increased, the number of request messages to be transmitted by the client also increased. As shown in Figure 16a, even if the number of groups was increased in the case of the DA-GW, the mean throughput was constant. This was because responses can be received from all group members within timeout (i.e., 60 ms) despite increasing the number of groups. In contrast, there were cases where all responses were not received from all group members within the timeout in existing methods as the number of groups was increased, thereby decreasing throughput. When the number of groups was three, a higher throughput was obtained when using the DA-GW than that of gateway-centric unicast by 73.72% and that of gateway-centric multicast by 24.93% approximately.

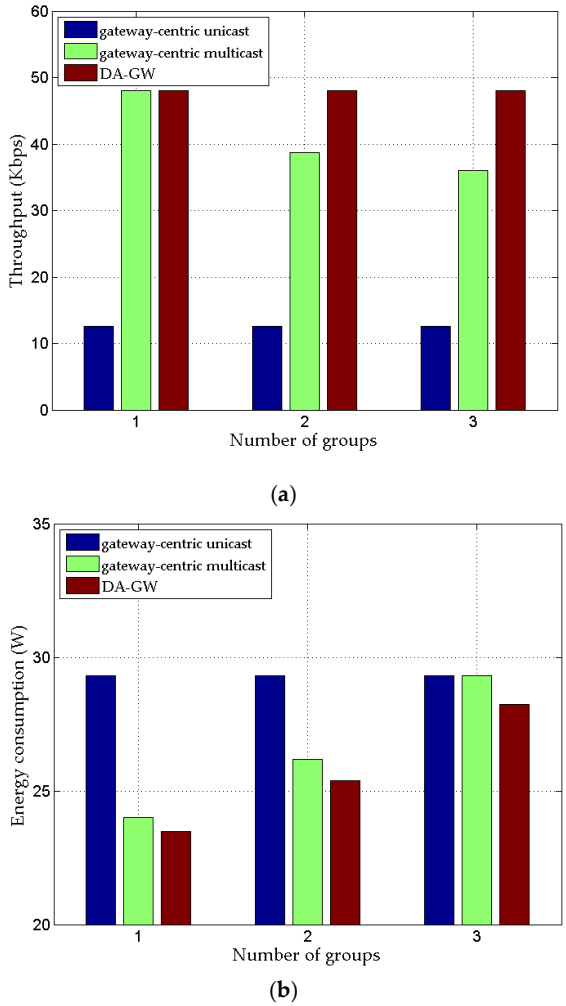


Figure 16. Multi-group experiment. (**a**) Throughput for a variation in the group size; (**b**) energy consumption for a variation in the group size.

Figure 16b shows the energy amount consumed by the client over 5 min according to increases in the number of groups. When the DA-GW was used, the energy consumption increased as the number of groups was increased. This was due to more request messages being transmitted than when the number of groups was fewer. When the DA-GW was used, the number of received response messages

was the least, indicating the lowest energy consumption. On the other hand, when gateway-centric unicast was used, constant energy consumption was revealed as the number of groups was increased. This was due to no changes in the transmission of request messages even if the number of groups was increased.

4. Conclusions

In this paper, a data aggregation gateway framework for CoAP group communications was proposed. The DA-GW consists of four functional blocks and three informative DBs. The functional blocks define the functions of the DA-GW, which consists of a message analyzer, group manager, message scheduler and data handler. The informative DBs store the resources for group communication, which consist of a client DB, resource DB and information DB. The DA-GW employs the oneM2M TS-0001-based data structure to provide interoperability for heterogeneous devices that use a variety of applications. The DA-GW was implemented using jCoAP to evaluate the functions and performance of the DA-GW. The measurement results on the performance of the DA-GW showed it had better performance than gateway-centric unicast by 73.71% and gateway-centric multicast by 19.47% in terms of throughput and had lower energy consumption than gateway-centric unicast by 13.37% and gateway-centric multicast by 2.98%, under the experimental conditions that the timeout period is 60 ms, the number of groups is two and the number of members in each group is 9.

Acknowledgments: This research was supported by the Leading Human Resource Training Program of Regional Neo Industry through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning (2016H1D5A1910427). This research was also supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2016-R0992-16-1006) supervised by the IITP (Institute for Information & communications Technology Promotion). This research was also supported by Hallym University Research Fund, 2016 (HRF-201605-009).

Author Contributions: Eui-Jik Kim conceived of and designed the overall framework. Minki Cha performed the open source-based implementation. Jung-Hyok Kwon implemented the resource structure. Minki Cha and Jung-Hyok Kwon conducted the experiment. Taeshik Shon and SungJin Kim interpreted and analyzed the data. Minki Cha, Jung-Hyok Kwon and Eui-Jik Kim wrote the paper. Eui-Jik Kim guided the research direction and supervised the entire research process.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. Internet of things (IoT): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.* **2013**, *29*, 1645–1660. [CrossRef]
2. Mainetti, L.; Patrono, L.; Vilei, A. Evolution of wireless sensor networks towards the Internet of things: A survey. In Proceedings of the 19th International Conference on Software, Telecommunications and Computer Networks, Split, Croatia, 15–17 September 2011; pp. 1–6.
3. Vermesan, O.; Friess, P. *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*, 1st ed.; River Publishers: Aalborg, Denmark, 2013; pp. 7–68.
4. Colitti, W.; Steenhaut, K.; Caro, N.D. Integrating wireless sensor networks with the Web. In Proceedings of the Workshop on Extending the Internet to Low Power and Lossy Networks, Chicago, IL, USA, 11 April 2011.
5. Shelby, Z.; Hartke, K.; Bormann, C. The constrained application protocol (CoAP) (RFC 7252). Available online: https://tools.ietf.org/html/rfc7252/ (accessed on 24 October 2016).
6. Kuladinithi, K.; Bergmann, O.; Pötsch, T.; Becker, M.; Görg, C. Implementation of CoAP and its application in transport logistics. In Proceedings of the Workshop on Extending the Internet to Low power and Lossy Networks, Chicago, IL, USA, 11 April 2011.
7. Richardson, L.; Ruby, S. *RESTful Web Services*, 1st ed.; O'Reilly Media: Sebastopol, CA, USA, 2007; pp. 1–22.
8. Rahman, A.; Dijk, E. Group communication for the constrained application protocol (CoAP) (RFC 7390). Available online: https://tools.ietf.org/html/rfc7390/ (accessed on 24 October 2016).
9. Konieczek, B.; Rethfeldt, M.; Golatowski, F.; Timmermann, D. Real-time communication for the Internet of things using jCoAP. In Proceedings of the IEEE 18th International Symposium on Real-Time Distributed Computing (ISORC), Auckland, New Zealand, 13–17 April 2015; pp. 134–141.

10. Law, L.K.; Krishnamurthy, S.V.; Faloutsos, M. Understanding the interactions between unicast and group communications sessions in ad hoc networks. In *Mobile and Wireless Communication Networks*, 1st ed.; Belding-Royer, E.M., Agha, K.A., Pujolle, G., Eds.; Springer US: New York, NY, USA, 2005; Volume 162, pp. 1–12.
11. Ishaq, I.; Hoebeke, J.; Abeele, F.V.D.; Rossey, J.; Moerman, I.; Demeester, P. Flexible unicast-based group communication for CoAP-enabled devices. *Sensors* **2014**, *14*, 9833–9877. [CrossRef] [PubMed]
12. Jung, M.; Kastner, W. Efficient group communication based on Web services for reliable control in wireless automation. In Proceedings of the IECON 2013-39th Annual Conference of the IEEE Industrial Electronics Society, Vienna, Australia, 10–13 November 2013; pp. 5716–5722.
13. Ishaq, I.; Hoebeke, J.; Rossey, J.; Poorter, E.D.; Moerman, I.; Demeester, P. Facilitating sensor deployment, discovery and resource access using embedded Web services. In Proceedings of the 2012 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, Palermo, Italy, 4–6 July 2012; pp. 717–724.
14. oneM2M. TS-0001-Functional-Architecture-V-2.10.0. Available online: http://www.onem2m.org/images/files/deliverables/Release2/TS-0001-%20Functional_Architecture-V2_10_0.pdf (accessed on 24 October 2016).
15. jCoAP. Available online: http://code.google.com/p/jcoap/ (accessed on 24 October 2016).
16. jCoAP open source. Available online: https://gitlab.amd.e-technik.uni-rostock.de/ws4d/jcoap (accessed on 24 October 2016).
17. Papathanasiou, C.; Tassiulas, L. Multicast transmission over IEEE 802.11n WLAN. In Proceedings of the IEEE International Conference on Communications, Beijing, China, 19–23 May 2008; pp. 4943–4947.
18. Selvam, T.; Srikanth, S. Performance study of IEEE 802.11n WLANs. In Proceedings of the 1st International Communication Systems and Networks and Workshops, Bangalore, India, 5–10 January 2009; pp. 1–6.
19. Introducing JSON. Available online: http://www.json.org/ (accessed on 24 October 2016).
20. Raspberry Pi 2 Model B. Available online: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/ (accessed on 24 October 2016).