ECHONET Lite-based IoT Platform for Smart Homes

Hoai Son Nguyen, Xuan Anh Do, Hoang Le, Van Hoang Nguyen, Quang Khai Duong, Xuan Viet Cuong Nguyen, Minh Hoang Ngo
Faculty of Information Technology
VNU University of Engineering and Technology, Vietnam National University, Hanoi
Hanoi, Vietnam
{sonnh,16020192, 16020229,16020242,16020208, 16020064}@vnu.edu.

***Abstract*— In this paper, we introduce an ECHONET Lite based IoT platform which allows households to set up smart home environment with ease as well as allows service providers to deploy their IoT applications easily and securely. Our proposed platform is fully implemented with three main features: a simple process for integrating new IoT devices into smart home environment, a seamless data transfer mechanism between IoT devices and service providers, and a RESTful API for verifying and supporting service providers to deploy their IoT applications into our platform. The experimental results verified the practicability of our proposed IoT platform.**

Keywords*— *IoT Platform, Smart home, ECHONET Lite protocol, MQTT protocol, JWT-based authentication

I. INTRODUCTION

The technological breakthroughs have facilitated daily life of people all around the world. One of the most successful innovations which improves standard of living is the Internet of Things (IoT). The simple conception of IoT is a network which allows all the things in the world to connect and exchange data with each other. When we bring IoT technologies to domestic space, we can introduce a new comfortable living environment called "smart home" in which various domestic services such as home automation, smart lighting, anomaly detection for IoT devices, smart air conditioner, etc. are provided to home users.

In a smart home, IoT devices collect and send data they obtain from the real world to service providers, who analyze the data and send control commands to IoT devices to provide services to home users. In fact, the increasingly popular of IoT devices and services for smart homes leads to a demand of an IoT platform which allows home users to easily set up smart home environment and allows service providers to easily access and control IoT devices inside a smart home. It means that the IoT platform needs to facilitate the interaction amongst IoT devices, households as well as service providers.

There are a large number of researches on the design of IoT platforms for smart homes. One design approach is cloud-based platform which utilizes cloud computing paradigms to achieve scalability, flexibility and high secure in data storage [1-4]. However, these platforms lack the ability of integrating various kinds of IoT devices to the system with ease. On the other hand, a number of IoT developers choose local platform to deploy their services [6-10]. Contrary to cloud-based platforms, local platforms handle all requests locally instead of pushing them into cloud. As a result, data transmission delay of local platforms is much smaller than cloud-based platforms. However, it is difficult to deploy applications which require large computing resources likely deep learning-based applications.

With the aim of allowing households to set up and manage their smart home environment with ease and supporting service providers deploying their IoT applications in smart home environment securely and efficiently, we propose in this paper the design of an ECHONET Lite-based IoT platform. First of all, our platform is implemented with ECHONET lite, a protocol designed for data communication in smart home [12]. Due to the flexibility of ECHONET Lite protocol, our IoT platform does not only enable smart devices' connectivity and management like other platforms, but also include some valuable features.

- A simple process for integrating new smart devices to the ECHONET Lite environment
- A seamless data transfer mechanism between IoT devices
- and service providers
- An API for service providers to easily deploy their services
- An JSON Web token-based authentication mechanism to authenticate users and specify service providers' access permission

In order to verify the practicability of our IoT platform, we implemented a testbed which is a smart lighting system with the use of our proposed IoT platform. The performance of our IoT platform is evaluated by the use of the testbed.

The paper is organized as follows. Section II presents past works in IoT platforms. Section III describes novelties of our platform in three part: Home Gateway, Server part and Application part. Section IV illustrates how we deploy our IoT platform and provide a method to evaluate our platform's performance. Finally, section V draws final conclusions.

II. RELATED WORKS

In an effort to work out feasible IoT solutions, the development of IoT platforms has attracted considerable attention. Particularly, two design approaches have been conducted for the construction of IoT platforms in recent years: cloud-based platforms and local platforms.

Cloud-based platforms are typical examples of the convergence between the IoT and the cloud computing paradigms. The authors of the paper [1] had proposed a platform based on a novel multi-layer cloud architectural model for smart homes, which substantially improves the interactions/inter-operations between heterogeneous home devices and services supplied by different vendors in IoT-based smart home. In comparison with the platform introduced in the paper [1], [2-4] are the prevalent ones that are not restricted to only smart home. Kaa [2] is a feature-rich open and efficient IoT cloud platform which supports various integrations for better scalability. Kaa platform empowers integration of not only a wide range of hardware types but

also different kinds of software such as data analytics, business tools and legacy systems. ThingSpeak – the platform proposed in the paper [3], distinguishes itself by generating a sense of community through the capability of creating public data-sharing channels. Furthermore, ThingSpeak offers users prominent features such as free hosting for channels, MATLAB visualization and additional support for the programming languages Ruby, Node.js and Python. Meanwhile, Mainflux [4] is another open-source, scalable and secure IoT platform based on the cloud. In addition to easy deployment and supplement of visualizations and analytical tools, Mainflux is capable of delivering over-the-air firmware updates and supporting all widely-used protocols - MQTT, WebSockets, CoAP and REST API [5].

While the above cloud-based platforms [2-4] have demonstrated their strengths, particularly overall cost and flexibility, there are great many things remaining to be done. Kaa IoT platform, in spite of its considerable potential, does not have many hardware modules supported and cannot deploy applications based on the PaaS model as well. With regard to ThingSpeak, that there is a limit when uploading data to the API and some required coding knowledge can be an obstacle to its users. In the meantime, Mainflux users need to host the IoT platform by themselves and are only able to exploit cloud services with commercial version.

In parallel with the blooming of cloud-based platforms, there exists abundant local IoT platforms which have data locally stored. OpenRemote [6] is an open-source project that focuses on home automation, commercial buildings, public spaces and healthcare. This platform allows users to integrate any device, protocol or design using available resources like iOS, Android or web browsers. Moreover, local data storage can truly offer the full data ownership to its end-users, thereby raising the level of user privacy. Thing Broker [7], another platform developed based on centralized architecture, delivers a Twitter-based abstraction model for things and events. The mission of Thing Broker is to integrate IoT objects with different characteristics, protocols, interfaces and constraints as well as to concurrently preserve the simplicity and flexibility of applications. Besides using a single abstraction to represent things, Thing Broker provides a model of relationship between things, which resembles the notions of "following" and "followed" links as observed in Twitter, to connect the world of heterogeneous entities that need to be accessed and controlled. [6] and [7] basically meet the major needs that customers expect from an IoT platform. However, local platforms require lots of efforts of service providers to deploy and maintain smart services in smart homes due to the lack of computing resource and storage resources.

ECHONET Lite [8] is an open communication protocol developed by the ECHONET Consortium which has turned out to be an international standard via ISO/IEC. The primary aim of ECHONET Lite is to develop a home network system that enable interconnectivity between multi-vendor electrical appliances, which are frequently found in ordinary houses. A remarkable feature of ECHONET is the use of lightweight messages which help to speed up transmission amongst IoT devices in a home network as well as avoid wasting transmission bandwidth and energy. Besides, the fact that ECHONET Lite protocol is supported by various IoT vendors as well as designed in object-oriented paradigm results in lower hardware cost and faster deployment. Upon characteristics, in ECHONET Lite protocol, communication between devices is carried out on a node-by-node basis, then there exists a term called ECHONET Lite node. A node is an entity on an ECHONET Lite network and uniquely identified by a single IP address. An ECHONET Lite node encompasses a profile object that stores information about the node and a device object list containing the names of one or more device objects. A device object is a logical model of the information held by sensors or home electrical appliances. Since device objects are classified based on types of devices, even products of different manufacturers that are of the same device type can be remotely controlled in the same way through a standardized interface form. However, ECHONET Lite only supports local communication methods and does not specify a design of a whole smart home system.

III. ECHONET LITE-BASED IOT PLATFORM

A. IoT platform design

In this paper, we propose the design of an IoT platform which allows households to easily deploy smart home systems and obtain the benefits of smart homes in small cost and allows service providers to easily deploy their applications on smart home environment. In order to do that, the IoT platform must deal with a number of following tasks.

- Connect and integrate multi-vendor smart IoT devices inside a smart home into the platform
- Collect and store sensor-actuator data for future use such as data analysis or data monitoring
- Allow only 3rd party service providers which receive households' permissions to access and send control commands to IoT devices
- Provide a simple and easy-to-use API for 3rd party service providers to deploy their applications

Our proposed IoT platform are structured into 3 main sides: Local side, Server side and User side (Fig. 1). Each side is assigned a separate task and builds on the component packaging mechanism that only one component of a side can connect to the another side.

The Local side is implemented inside a smart home. It contains one Home Gateway and a number of smart IoT devices. Its mission is to collect sensor data about the home's living environment and execute control commands sent by service providers on actuator devices.

Smart IoT devices include 2 types:

- Sensor devices: measure physical parameters such as temperature, humidity, illuminance, electrical current, etc. of home environment
- Actuator devices: change the home's environment to meet users' requirements (E.g. light bulbs, HVAC devices, curtain controllers, etc.).

We utilize ECHONET Lite as a communication protocol between smart IoT devices and Home Gateway. Each IoT device is considered as an ECHONET Lite device. By the use of ECHONET Lite protocol, our IoT platform can easily connect ECHONET Lite compatible devices of different vendors and perform data transmission between smart IoT devices and Home Gateway in small cost.

Home Gateway is a mini computer that serves as a communication bridge between smart IoT devices and Server side. Home Gateway takes responsibility for four following

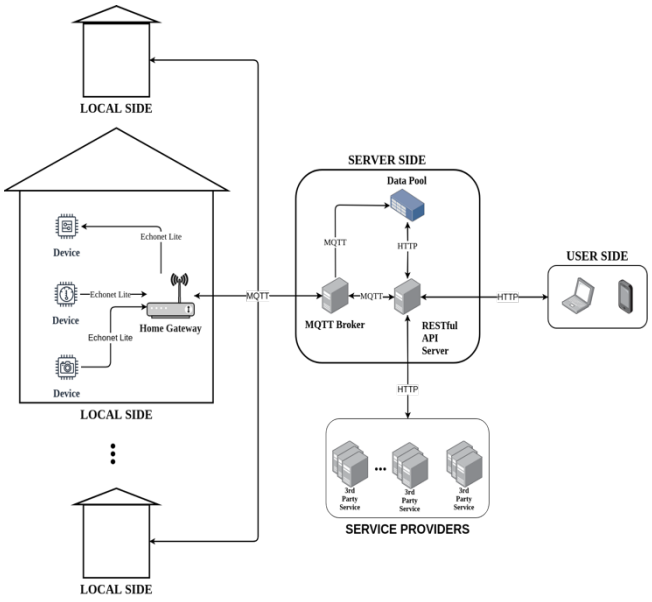

Fig. 1 Overview of ECHONET lite-based IoT platform

operations: identifying all smart IoT devices already joined in the network and detecting new devices, monitoring and controlling devices and exchanging data with Server side. The Home Gateway is the only component that is capable of communicating with the Server side, which helps to easily manage and control smart IoT devices.

The Server side is implemented in physical servers in cloud. Server side is structured into 3 components: MQTT broker, Data pool and RESTful API server.

The first component is MQTT Broker (Broker) whose main job is to support data exchange between Local side and Server side using MQTT protocol [13]. The usage of MQTT protocol brings to our platform many advantages such as highly decoupled publisher and subscriber model, support of asynchronous messaging and 3-level QoS.

Our IoT platform stores collected actuator-sensor data in Data pool, the second component of the Server side. The Data pool also stores information of home users' account, home information, service providers' information and other information necessary for managing all activities in smart home.

The third component is RESTful API server. In our design, RESTful API server takes an important role of providing interface for service providers to access data and send control commands to smart IoT devices inside a smart home. The location of RESTful API server enables the IoT platform to control the information provided to service providers and check the commands sent by Service providers to secure user's information as well as their safety. RESTful API server authenticates users and service providers by the use of JSON Web token to ensure their access privileges.

We design the architecture of the Server side according to the idea of a black box in which only MQTT Broker has the ability to communicate with Local service and only API server can communicate with End-User side. This design helps the system to be secured, easy to maintain, change and increase system scalability.

The household of a smart home can monitor and control IoT devices inside the smart home by the use of an application provided by User side. The User side can provide a web application or a mobile application, which may be installed on households' smart phones.

B. Device naming

In Local side and Server side we use different naming schemes to flexibly specify the smart IoT devices that the system needs to access and control. In Local side, we need a naming scheme which is simple enough to implement and save transmission cost. In Server side, we need a naming scheme which can allow service provider to access to the right smart IoT device inside a smart home in an easy and flexible way.

On the Local side, an IoT device is identified by a device ID, which is a combination of its IP address of the ECHONET lite node it belongs to and EOJ code which specifies the type of the IoT device. EOJ code includes group code, class code and instance code, which are defined by ECHONET lite standard. The group code and the class code written in hex code play the role as an identifier for a particular group of devices and a specific device type in that group. For example, an air pollution sensor has a class code of 0x0B, an illuminance sensor has a class code of 0x0D and an electric energy sensor has a class code of 0x22. These sensors have the group code of 0x00.

Meanwhile, on the Server side, device name is set according to a hierarchical naming scheme that includes five levels, namely: "user ID", "home ID", "room name", "device type", and "device ID". The attribute "user ID" specifies the user account that is granted to the household of one or more smart homes. The attribute "home ID" is an identification which identifies a smart home of the household. In fact, a person can own several smart homes. The attribute "room ID" indicates a place (outside, bedroom, living room, kitchen, etc.) in which the device is installed. The attribute "device type" identifies the type of the device, which corresponds to the group code and class code defined by ECHONET lite standard. Finally, "device ID" specify an ECHONET lite device by a combination of the physical address (e.g. MAC address) of ECHONET lite node and the instance code of the ECHONET lite device. The physical address is used in order to identify the ECHONET lite node while the instance code is necessary in the case multiple ECHONET lite devices exists on a single ECHONET lite node.

We use a hierarchical naming scheme on the Server side since the naming scheme allows service providers to flexibly and easily describe the information that they need. For example, the naming scheme allows a service provider to get information of all devices in a room or information related to a device type in a room by sending one request. They can also exactly specify the device that they want to access and control.

C. Integration of smart devices into IoT platform

In our proposed IoT platform, a new ECHONET Lite device will be automatically integrated into the platform after joining home network based on the ECHONET Lite protocol standard. When an ECHONET Lite device joins the home network, it will be automatically discovered by IoT platform and an appropriate monitoring scenario is set up to obtain device data and send it to the Server part. Control commands sent from service providers will also be delivered to the right devices without loss.

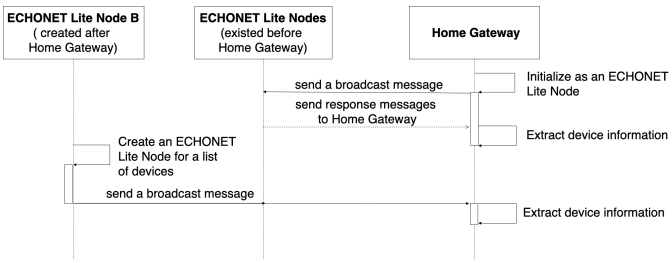

Fig. 2 Device discovery process

The discovery of devices in the Local side of our IoT platform is realized through a broadcast channel defined by ECHONET Lite protocol. As shown in Fig. 2, once entering the home network, Home Gateway will send a notification request message to retrieve all existing nodes and its devices' information using the broadcast channel. All ECHONET Lite nodes that previously exist in the home network can receive this message and send a response message containing a device object list and a profile object to Home Gateway. Home Gateway then establishes listening events to detect new devices whenever their ECHONET Lite node participates in the network. A newly joined ECHONET Lite node also sends a notification message with a device object list and a profile object on the broadcast channel as soon as it joins the home network. This message will be delivered to all ECHONET Lite nodes including Home Gateway and be caught by a corresponding listening event at Home Gateway. Based on ECHONET Lite specification in [12], Home Gateway will identify which device that a device object corresponds to.

Regarding the timely delivery of data to service providers to analyze and take appropriate actions, each time a device is discovered, a corresponding monitoring scenario will be set based on the type of the device. Devices such as actuators only need to send update data in case there are any status changes but sensors may need send their data periodically. By sending a GET request message with appropriate data fields to a specific device in a period of time, Home Gateway can easily get status information of the device at a predetermined period. Besides, an ECHONET Lite node can notify other nodes including Home Gateway in case its devices have changes in terms of operation status by sending notification messages on the multicast channel. Whenever receiving a status update message, Home Gateway will convert the content of the message to JSON format and send the data to the Server side (i.e. MQTT broker). Details of updating data in the Server side will be described in Section 3.4.

Our IoT platform will deliver a control command sent by a service provider to a specific ECHONET Lite device in the home network in order to control the device for service provision. Whenever Home Gateway receives a message containing a control command from Server side, it converts the command into ECHONET lite format and sends a SET request message to the device. The ECHONET Lite node, upon receiving the message, will change its own devices' status and send a response message to notify that its devices' status has been updated successfully.

Since a control message is essential for service provision, our IoT platform guarantees the delivery of the SET request message. Since the message can be lost during data transmission in the home network, Home Gateway will set a

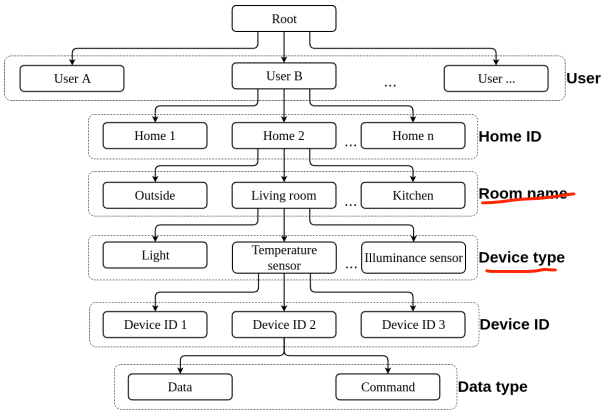

Fig. 3 MQTT topics for IoT devices

timer after sending the message. In case a SET request message is lost, the receiver will not receive a response message within the predefined timeout. In this case, the Home Gateway will resend the SET request message to the receiver until it receives a response message indicating that device status is updated successfully.

D. MQTT-based interaction within Server side

In order to send data between components within Server side and between Local side and Server side we utilize MQTT protocol. There are three main data flows in the IoT platform:

- Home Gateway publishes sensor-actuator data to MQTT broker which forward the data to Data poll and API servers which require the data.
- API server publish control commands destined to an IoT device inside a smart home to MQTT broker, which will forward the command to the Home gateway of the smart home.
- API server request data from Data pool.

Since MQTT protocol utilize topics to determine the receiver of published data, we organized MQTT topic based on device names. For each device name, we have two topics: one topic for data publication from the device and one topic for sending control commands from Server side to the device (Fig. 3).

Sensor-actuator data in all smart homes will be stored in Data pool for home monitoring and data analysis. In order to do that, the Data pool acts as a MQTT subscriber, which subscribes the root topic on MQTT broker. All data sent to MQTT broker will be forwarded to Data pool to store. We also store information about user's account and their smart home's structure in Data pool.

Beside fundamental services provided by IoT platform, most of smart home services for households will be provided by Service providers via RESTful API servers. Service providers may require to receive two types of data: real-time data and historical data. In order to provide real-time data of IoT devices to service provider, RESTful API server subscribes to the topic that the data is published at MQTT broker. For example, if the service provider request the temperature data of the living room of the home "Home 2" of user B, the API server will subscribe a MQTT topic of "/userB/home2/living room/temperature sensor/*" at MQTT broker. If a service provider requests historical data of an IoT device, the RESTful API server will send the request to Data pool and forward the received data to Service provider.

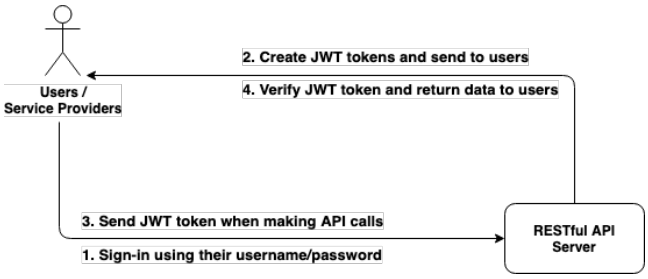

Fig. 4 Authentication mechanism using JWT

In order for a service provider to send control commands to actuator devices in a smart home, the Home gateway will subscribe topics corresponding to actuator devices it is responsible for. When a control command destined to an IoT device is sent to RESTful API server, the API server will check the validity of the command and publish the command to the topic corresponding to the IoT device, which will be forwarded to the responsible Home gateway.

E. JWT-based authentication mechanism

For a smart home, access control to IoT devices is very important. In order to receive sensor/actuator data and send control commands to devices within a smart home, service providers must obtain user approval on which devices they can access depending on the service they provide.

In our IoT Platform, we define 3 user types based on data access levels: Administrator, normal user and service provider. The administrator has full access privileges to the system. The normal user who owns a smart home has full access privileges to devices inside the smart home that he/she owns. Service providers can only access and control devices which are approved by the user.

We design an authentication mechanism to verify the access privileges for normal users and service providers (or users in brief), using JSON Web Token (JWT) [14], a light-weight authentication mechanism which can create a truly RESTful API without session maintenance after logging in. JWT has 3 parts: Header, Payload and Signature. Header part stores information about the token type (i.e. JWT in this case) and the algorithm used to create the token. Payload part contains user ID and access privileges of the authenticated user. Here, the access privileges are the list of MQTT topics that the user can access and the time-to-live of each topic. Signature part is created from Header part, Payload part and a secret key by the use of the algorithm specified in Header.

RESTful API server utilizes JWT token to verify the access privileges of users each time it receives a data request (Fig. 4). When a user logs in the system with their username and password, the API server returns an initial JWT token to the user. When the user sends an access request for a smart home along with the JWT token, the API server will check the MQTT topic corresponding to the access request. If the topic is included in the payload of the JWT token, the access request is valid and the API server will execute the request. If the topic is not included in the payload of the JWT token, the API server will send a request to Data pool to get the access privilege of the user. If the access request is valid, the API server will execute the request, as well as update the list of MQTT topics in the payload part of the JWT token and send back to the user.

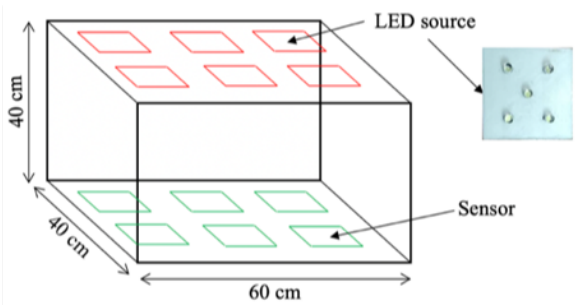

Fig. 5 A prototype of a smart lighting system

In order to reduce the size of JWT token and the overhead and delay caused by Data pool access of RESTful API server, when updating the list of MQTT topics in the payload part of the JWT token, the highest-level MQTT topic that the user can access is used. For example, if a user request corresponds to a MQTT topic of "/userB/home2/living room/temperature sensor/device ID1/data" but the highest topic that the user can access is "/userB/home2/living room/*" (i.e. the user can access to all devices in the living room), the topic will be used to update the JWT token. Further, we limit the topic number in the list of MQTT topics. If the topic number is over the maximum number, the old topic will be deleted.

IV. IMPLEMENTATION AND EVALUATION

A. Installation environment

We implement our IoT platform in a testbed environment in order to verify our design and evaluate the performance of the IoT platform in three components: local side, server side and user side.

In the Local side, we used OpenEcho library [15] to implement sketches for the Home Gateway because it has pre-installed abstract classes, along with the methods and attributes which support specifically for ECHONET Lite protocol.

In the Server side, we used Mosquitto open source software to implement MQTT Broker. In regard to the database for Data pool component, we employed MySQL, which is the most popular open source database provided by Oracle.

For the User side, we develop a web application using Node.js and an iOS mobile application. Both mobile and web applications are configured to the API server with the aim of enabling users to control and manage their connected household devices in a convenient way.

B. Build a smart lighting system based on IoT Platform

To prove the feasibility of the IoT platform that we have designed above, we have built a smart lighting control service that automatically turn on/off lights to match the user's desired brightness while minimizing the power consumption of the whole system as much as possible.

The smart lighting control service provider uses our proposed IoT platform to deploy its service. When a user sends his/her desired brightness in some areas in a room to the smart lighting server by the use of a mobile application, the server interacts with RESTful API server to get illuminance data from IoT devices in the room such as lighting illuminance or nature illuminance. Then, the smart lighting server calculates an optimal light turning pattern

based on the illuminance data and users' desired brightness and sends back commands via API server to control lights in the room.

The system is integrated with Genetic Algorithm (GA), which helps our system optimize light turning pattern to meet user's desired brightness. It also maximizes the energy saving by utilizing natural illuminance efficiently.

In order to verify the operation of our implemented smart lighting control service, we built a plastic home prototype (Fig. 5). The home prototype has 6 smart lighting kits and 6 illuminance sensors which are controlled by 6 ESP8266 Wi-Fi modules. We use BH1750 illuminance sensors to measure illuminance value at each lighting area. In addition, we installed the source code of Home Gateway into a Raspberry Pi, which is running on Raspbian OS. We implemented Data pool, RESTful API Server and MQTT Broker on a server running on Ubuntu 18.04 with a 10-core CPU.

The experimental results of the prototype system have proved that the smart lighting system worked properly as expected. The lighting devices and illuminance sensors were automatically integrated into the system and the lighting control service provider could obtain illuminance sensor data and sent control commands to the lighting devices. It does not only bring convenience to the user, but also helps saving energy of the whole indoor lighting system brightness.

C. *Perfomance evaluation*

To evaluate the performance of the smart lighting system built on our IoT platform, we conducted an experiment of a lighting control scenario and measure the delay when controlling the lights. In this experiment scenario, the user set the required brightness level at a location inside the home prototype. The experimental results are denoted in Table 1.

TABLE 1. DELAY TIME WHEN SENDING MESSAGES IN THREE SECTIONS

Section	Delay time
Mobile application sends request to service provider (i.e. lighting controller)	15ms
Service provider runs algorithm to result in light turning pattern	300ms
Service provider sends requests to Home Gateway	4ms
Home Gateway sends a command until a lighting device is controlled at right level	1.4s – 1.7s

Table 1 illustrates the delay time in four message transmission sections in our IoT platform. The delay between Home Gateway and lighting devices is the largest in four sections. It is mainly because of the processing time at the smart lighting kits, which are processed by ESP8266 Wi-Fi modules. The running time of GA-based optimization algorithm at the service provider also contribute large delay of the system. Other delays are small enough comparing to these two delays.

V. CONCLUSION

In this paper, we have introduced an IoT platform which enables users to set up and manage smart home environment with ease as well as allows approved service providers to access IoT devices in order to develop IoT services in smart homes. ECHONET Lite protocol is the key factor of our platform when compared to myriad other IoT platforms due to its light-weight messages, multi-vendor support and object-oriented device modeling. Besides, our platform provides a simple process for integrating new IoT devices into the ECHONET Lite environment, equips reliable transmission services at Home Gateway and supplies an API for service providers to receive device access permission and deploy their IoT services easily. Based on the process of development and experiments undertaken, our ECHONET Lite-based platform has proven its flexibility and stability, which can leverage the large-scale deployment.

In our future work, we plan to further develop basic IoT applications such as devices' data visualization and improve the performance of our proposed IoT platform.

REFERENCES

[1] Tao, M., Zuo, J., Liu, Z., Castiglione, A., & Palmieri, F. (2018). Multi-layer cloud architectural model and ontology-based security service framework for IoT-based smart homes. Future Generation Computer Systems, 78, 1040-1051.

[2] Kaa IoT platform. Retrieved September 2, 2019, from https://www.kaaproject.org/

[3] Maureira, M. A. G., Oldenhof, D., & Teernstra, L. (2011). ThingSpeak–an API and Web Service for the Internet of Things. World Wide Web.

[4] Mainflux - Full-stack Open-Source, Patent-free IoT Platform and Consulting services. Retrieved September 2, 2019, from https://www.mainflux.com/

[5] The Top Open Source IoT Platforms for Developers. Retrieved September 2, 2019, from https://opensourceforu.com/2018/10/the-top-open-source-iot-platforms-for-developers/

[6] OpenRemote. Retrieved September 2, 2019, from http://www.openremote.com/

[7] Perez de Almeida, R. A., Blackstock, M., Lea, R., Calderon, R., do Prado, A. F., & Guardia, H. C. (2013, September). Thing broker: a twitter for things. In Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication (pp. 1545-1554), ACM.

[8] Soldatos, J., Kefalakis, N., Hauswirth, M., Serrano, M., Calbimonte, J. P., Riahi, M., ... & Skorin-Kapov, L. (2015). Openiot: Open source internet-of-things in the cloud. In Interoperability and open-source solutions for the internet of things (pp. 13-25). Springer, Cham.

[9] Medvedev, A., Hassani, A., Zaslavsky, A., Jayaraman, P. P., Indrawan-Santiago, M., Haghighi, P. D., & Ling, S. (2016, November). Data ingestion and storage performance of IoT platforms: study of OpenIoT. In International Workshop on Interoperability and Open-Source Solutions (pp. 141-157). Springer, Cham.

[10] Kostelnik, P., Sarnovsk, M., & Furdik, K. (2011). The semantic middleware for networked embedded systems applied in the internet of things and services domain. Scalable Computing: Practice and Experience, 12(3), 307-316.

[11] da Cruz, M. A., Rodrigues, J. J., Sangaiah, A. K., Al-Muhtadi, J., & Korotaev, V. (2018). Performance evaluation of IoT middleware. Journal of Network and Computer Applications, 109, 53-65.

[12] ECHONET Lite Specification. Retrieved September 2, 2019, from https://ECHONET.jp/spec_v113_lite_en/

[13] MQTT Protocol. Retrieved September 2, 2019, from http://mqtt.org/

[14] Json Web Token. Retrieved September 2, 2019, from https://jwt.io/

[15] OpenEcho. Retrieved September 2, 2019, from https://github.com/SonyCSL/OpenECHO/