Room Temperature Control and Fire Alarm/Suppression IoT Service using MQTT on AWS

N. Manasa[1], C. R. Meghana[2], S. Kavyashree[3], M. N. Meghana[4]
[1,2,3,4]*U.G. Student, Department of Electronics and Communication, BGSIT, Mandya, India*

***Abstract*—In this paper we build an MQTT (Message Queue Telemetry Transportation) broker on Amazon Web Service (AWS). The MQTT broker has been utilized as a platform to provide the Internet of Things (IoT) services which monitor and control room temperatures, and sense, alarm, and suppress fire. Arduino was used as the IoT end device connecting sensors and actuators to the platform via Wi-Fi channel. We created smart home scenario and designed IoT massages satisfying the scenario requirement. We also implemented the smart some system in hardware and software, and verified the system operation. We show that MQTT and AWS are good technical candidates for small IoT business applications.**

Index Terms— AWS, IoT, MQTT, Smart Home

I. Introduction

Even three or four years ago we did not dream that the IoT would come into our life so early. From GE to Belkin to Home Depot, tons of products and whole ecosystems want to help you control your home via a single iOS or Android app [1].

Now-a-days the IoT is becoming a novel paradigm that is rapidly gaining business area in the modern wireless telecommunications with the integration of several technologies and communications solutions.

MQTT is a publish/subscribe message exchange protocol developed by IBM [2]. The MQTT system consists of MQTT broker and client. The MQTT broker is a message exchange platform that enables the message producer client to publish messages with a message identifier Topic. When the message consumer client subscribes to the Topic, the MQTT broker delivers the topic messages. Recently, MQTT has been adopted as the message transfer binding protocol in oneM2M IoT international standards [3].

AWS offers a suite of cloud-computing services that make up an on-demand computing platform. As of 2016 AWS has more than 70 services, spanning a wide range, including compute, storage, networking, database, analytics, application services, deployment, management, mobile, developer tools and tools for the Internet of things. AWS is very attractive for small IoT business applications because they provide large computing capacity quicker and cheaper than a client company building an actual physical server farm.

In this paper we demonstrate that MQTT and AWS are good technical candidates for small IoT business applications. We created a smart home scenario and designed IoT massages satisfying the scenario requirements.

We also implemented the smart some system in hardware and software and verified the system operation. We build an MQTT broker on AWS. The MQTT broker has been utilized as a platform to provide the IoT services which monitor and control room temperatures, and senses, alarms, and suppress fire. Arduino was used as the IoT end device connecting a room temperature sensor, a fire sensor, and fire alarm, an air conditioner, and a sprinkler actuators to the platform via Wi-Fi channel. We used the Gluon mobile API for the development of mobile application. Application provides indoor temperature monitoring, desired temperature setting, fire alarm reception and suppression functions.

II. MQTT Protocol

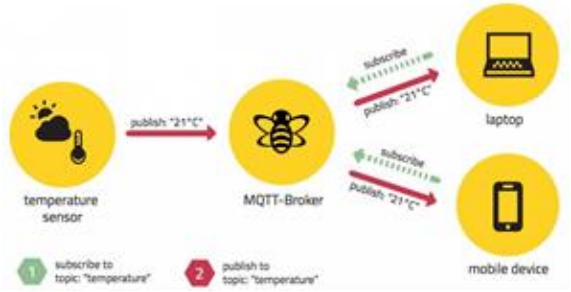


Fig. 1. MQTT protocol operation

Fig. 1 shows the MQTT protocol operation. The basic concepts of it is publish/subscribe and client/broker and its basic functionality is connect, publish, and subscribe. Also it has several good features like quality of service, retained messages, persistent session, last will and testament and SYS topics. MQTT decouples the space of publisher and subscriber. So they just have to know hostname/ip and port of the MQTT decouples the space of publisher and subscriber. So they just have to know hostname/ip and port of the broker in order to publish/subscribe to messages. The broker is able to store messages for clients that are not online. MQTT is also able to decouple the synchronization, because most client libraries are working asynchronously and are based on callbacks or similar model. So it won't block other tasks while waiting for a message or publishing a message. But some libraries have synchronous APIs in order to wait for a certain message.

MQTT is really the essence of pub/sub when using a client library and that makes it a light-weight protocol for small and constrained devices.

III. Smart Home IoT Design

1) Scenario:

In the section we describe our smart home IoT scenario for

the reference implementation with MQTT on AWS.

In the room, the cooling/heating unit (air conditioner) and the temperature sensor are interlocked so that the room temperature can be controlled automatically. Fire alarm and suppression are possible by installing fire detection sensor and sprinkler. In the smartphone app, the room temperature can be monitored remotely regardless of location. If a desired temperature is set on the app, the app sends the target value via IoT platform to the temperature control system. Then the local end device controls the air conditioner so that the desired temperature is maintained.

Flame sensor, fire alarm, and sprinkler are also interlocked to provide automatic fire detection, alarming, and suppression service. When the flame sensor detects the fire, the IoT device automatically alarms the fire event, activates the sprinkler to suppress the fire, and sends fire alarm message to smartphone app via IoT platform

The user of the app checks the fire suppression message and checks the operation status of the sprinkler and forcibly operates the sprinkler when not in operation.

2) IoT message design based on MQTT protocol:

In the section we describe IoT message design based on MQTT protocol so as to suitable for the implementation of the above-mentioned smart home IoT scenario.

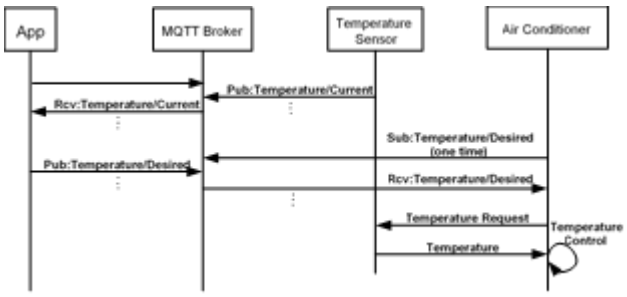
The Table-1 shows the IoT message for MQTT protocol operation.

TABLE I
IOT MESSAGE FOR MQTT PROTOCOL OPERATION

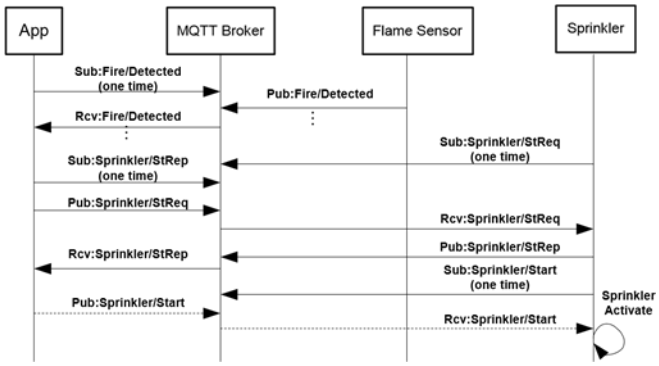
IoT Message	MQTT Topic	Publisher	Subscriber	Subscriber's Action
Current Tempera-ture	Temperature/ Current	Temperature Sensor	App	Display
Desired Temperature	Temperature/ Desired	App	Air Conditioner	Temp Control
Fire Detection	Fire/ Detected	Flame Sensor	App	Pop Up
Sprinkler Request	Sprinkler/ StReq	App	Sprinkler	Read Status
Sprinkler Reply	Sprinkler/ StRep	Sprinkler	App	Send Message
Sprinkler Start	Sprinkler/ Start	App	Sprinkler	Activate
System Request	System/ StReq	App	Devices	Send Message
System Reply	System/ StRep	Devices	App	Display

The IoT messages are for room temperature control, fire detection/suppression, and system status monitoring. For each message the name of MQTT topic was given, who will publish out and subscribe in was identified, and the subscriber's action after receiving the message was also defined.

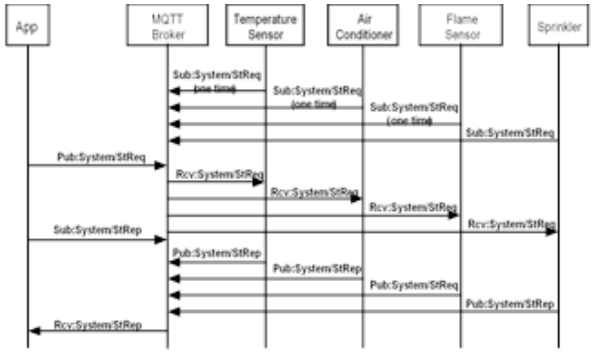
The Fig. 2 shows the IoT message exchange procedure according to MQTT protocol for room temperature monitoring/controlling, fire sensing/alarming/suppression, and system status checking.



(a) Temperature control



(b) Fire detection/suppression



(c) System checking

Fig. 2. IoT message exchange procedure

3) Smart home IoT design:

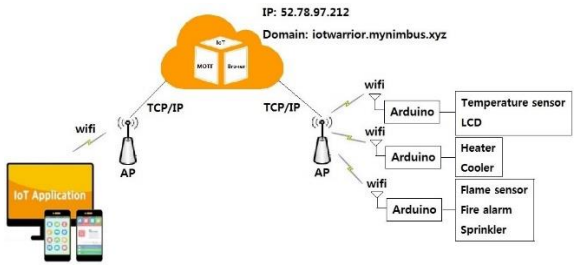


Fig. 3. Smart home IoT system architecture

The Fig. 3, shows the smart home IoT architecture that implements automatic temperature control and fire alarm/suppression. Three Arduino's are used. Two were used for temperature sensor and air conditioner respectively and one was used for fire alarm flame sensor, and sprinkler. Arduino's

are equipped with Wi-Fi module to communicate with MQTT broker. MQTT broker will be installed on AWS. Smart phone application will monitor the room temperature, control temperature, get fire alarm message on fire event, check sprinkler status, and active the sprinkler when not in action on fire event.

IV. IMPLEMENTATION

1) Implementing the MQTT Broker in AWS:

The Fig. 4 shows the MQTT Broker architecture built on AWS. The implementation procedures are as follow:

First, create an AWS account. Second, create a server using Elastic Compute Cloud (EC2) known as the most basic and widely used infrastructure in AWS, which provides a virtual server connected to the Internet. AWS automatically assigned public IP address 52.78.97.212 to our server. If customers want a permanent public IP address, they should have to pay cost. Otherwise, for every start of the server, new public IP address are assigned.

Third, connect to the public IP of the server with virtual terminal program such as putty and upload the MQTT broker to create IoT platform on AWS server.

Several types of brokers are available. This paper used a broker called Mosquitto [5]. The reasons why we use this broker are that it is an open source, easy to install and suitable for IoT.

For the easy access to the server, we subscribed in a commercial DNS service and mapped the server IP address to the URL of iot warrior.mynimbus.xyz. However, since the public IP address of AWS changes every time the server is rebooted, the new IP address must be re-mapped to the URL. The Broker receives the message on TCP port 1883. MQTT Topic are commonly available as "IoT Warrior / MQTT Topic" at tcp: //Iotwarrior.mynimbus.xyz:1883.

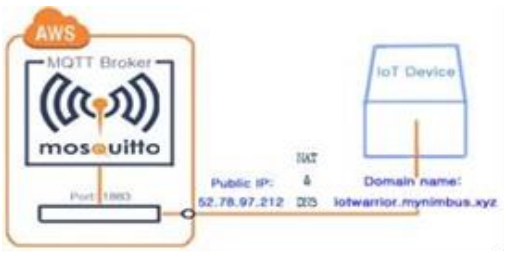


Fig. 4. MQTT Broker Architecture Built on AWS

2) Software implementing in smart phone app and end device:

Because the smartphone app, Broker, and Arduino require bidirectional communication, MQTT server and client functions must be implemented in each device. Followings are development tools and libraries for Arduino end device and Android smart phone.

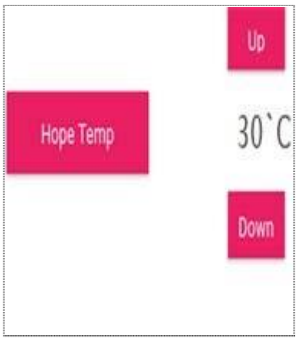
A) Arduino part implementation:

- *Development Tools and Languages:*
  Arduino IDE[6], C/C++ [7]
- *Libraries :*
  PubSubClient [7], WiFi Esp [9],
  LiquidCrystal_I2C [10],
  DHT_sensor_library [11]

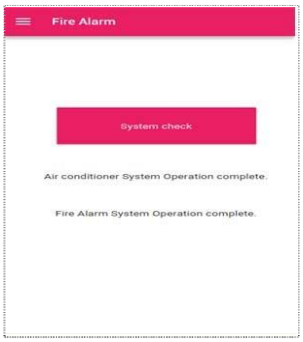
(B) App part implementation:

- *Development Tools and Language :*
  Gluon Plugin [12] in Netbeans [13]
- *JAVA Libraries :*
  charm-common-3.0.0 [14]
  paho.client.mqttv3 [15]

(a) Room temperature monitoring



(b) Desired temperature setting



(c) System Checking

(d) Received fire alarm

Fig. 5. Smart phone application for smart home IoT service.

The Fig. 5 shows screen shots of the smart phone application

for smart home IoT service. From (a) to (d) each figure shows room temperature monitoring, desired temperature setting, system checking, fire alarm message reception.

3) Hardware implementation for smart home IoT service:

The Fig. 6 shows an hardware implementation for smart home IoT service. The Arduino's of this hardware have Wi-Fi module to connect to MQTT broker in AWS. Smart Phone App also connect to the broker via Wi-Fi channel or 4G/5G web service. Because the broker has the public IP address, the mobile app can access to this hardware with the help of the broker from anywhere and anytime.

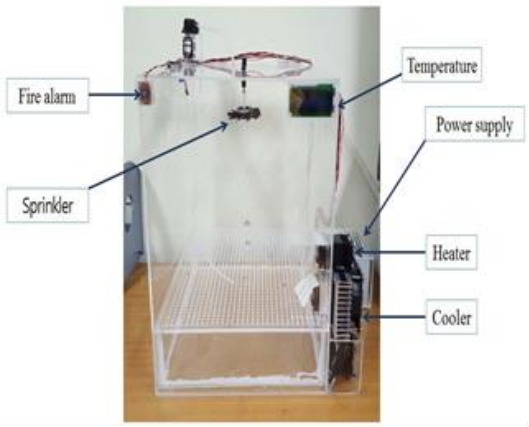


Fig. 6. Smart home hardware

4) System Experiments:

We made experiments on our smart home system.

A short video source is available on YouTube URL https://youtu.be/tHoDcmvezqA.

V. Conclusion

In this paper, we created a smart home scenario and designed IoT massages satisfying the scenario requirements. We also implemented the smart some system in hardware and software and verified the system operation. MQTT broker has been built in AWS. Utilized it as a smart home IoT platform to build a room temperature control and fire alarm/suppression system. With AWS, global access to IoT services is possible and server maintenance difficulties can be eliminated. In particular, global access is possible without separately providing public IP, making it well suited for individual or small business IoT service establishment. Based on the results of this study, we can conclude that MQTT and AWS are good technical candidates for small IoT business applications. Future research subjects are implementation of security services such as authentication and authorization, and extension of research area to oneM2M platform implementation.

VI. Future Work

The effective utilization of a publish/subscribe architecture of MQTT in the field of warehouse automation. The system proposed can be effectively utilized in the public warehouses wherein each of utilizes more bandwidth and has high latency than COAP when amount of application data being transferred is considered. On the other hand when it comes to the features such as architecture, ease of implementation, data integrity and security TCP based MQTT protocol always proves to be more efficient to use than COAP. The threat model for a typical IoT environment with particular emphasis on MQTT-based IoT deployments. We conducted experiments to quantify the impact of DoS attacks against the MQTT broker, with obtained results providing us with insight into the problem domain. As part of our future work we shall assess the impact of other malicious attacks on IoT devices and MQTT message brokers. In addition, the performance of a load-balanced MQTT broker environment during different attacks needs to be evaluated.

There is an urgent need to protect the IoT devices from malicious attacks and misuse which could impede the evolution of IoT as a reliable and secure paradigm. New research initiatives are required to detect attacks that target IoT devices and characterise their behaviour. Such initiatives will help build effective solutions that will mitigate and thwart such attacks.

A conclusion section is not required. Although a conclusion may review the main points of the paper, do not replicate the abstract as the conclusion. A conclusion might elaborate on the importance of the work or suggest applications and extensions.

References

[1] L Atzori, A. Iera and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.
[2] MQTT Web page. Source: http://mqtt.org.
[3] oneM2M, *oneM2M Release 2 specifications* - TS0010, ver. 2.41, Aug. 2016.
[4] MQTT Essentials. Source: http://www.hivemq. com/mqtt-essentials/
[5] Mosquitto Web page. Source: https://mosqu itto.org/
[6] Arduino Web page. Source: https://www.arduino.cc.
[7] Arduino Language.
Source: https://www.arduino.cc/en/Reference/HomePage.
[8] Arduino Client for MQTT. Source: http://pubsubcl ient.knolleary.net/
[9] WiFiEsp. Source: https://github.com/ bportaluri/WiFiEsp
[10] LiquidCrystal_I2C.
Source: https://github.com/marc oschwartz/LiquidCrystal_I2C.
[11] DHT_sensor_library.
Source: https: //github.com/adafruit/DHT-sensor-library
[12] NetBeans IDE. Source: https://netbeans.org/
[13] Gluon Mobile. Source: http://docs.gluonhq. com/charm/3.0.0/
[14] charm 3.0.0 API.
Source: http://docs.gluonhq. com/charm/javadoc/3.0.0/.
[15] Eclipse paho. Source: https://eclipse.org/paho/