International Journal of Electrical and Computer Engineering (IJECE)
Vol. 8, No. 5, October 2018, pp. 3784~3792
ISSN: 2088-8708, DOI: 10.11591/ijece.v8i5.pp3784-3792

An Event-based Middleware for Syntactical Interoperability in Internet of Things

Eko Sakti Pramukantoro, HusnulAnwari
Faculty of Computer Science, Universityof Brawijaya, Indonesia

Article Info

Article history:

Received Oct 10, 2017
Revised Mar 21, 2018
Accepted Apr 3, 2018

Keyword:

Actor based
IoT middleware
Message broker
Pub-sub
Web-oriented

ABSTRACT

Internet of Things (IoT) connecting sensors or devices that record physical observations of the environment and a variety of applications or other Internet services. Along with the increasing number and diversity of devices connected, there arises a problem called interoperability. One type of interoperability is syntactical interoperability, where the IoT should be able to connect all devices through various data protocols. Based on this problem, we proposed a middleware that capable of supporting interoperability by providing a multi-protocol gateway between COAP, MQTT, and WebSocket. This middleware is developed using event-based architecture by implementing publish-subscribe pattern. We also developed a system to test the performance of middleware in terms of success rate and delay delivery of data. The system consists of temperature and humidity sensors using COAP and MQTT as a publisher and web application using WebSocket as a subscriber. The results for data transmission, either from sensors or MQTT COAP has a success rate above 90%, the average delay delivery of data from sensors COAP and MQTT below 1 second, for packet loss rate varied between 0% - 25%. The interoperability testing has been done using Interoperability assessment methodology and found out that ours is qualified.

Corresponding Author:

Eko Sakti Pramukantoro,
Faculty of computer science,
University of Brawijaya,
8 Veteran Road, Malang, Indonesia - 65145.
Email: ekosakti@ub.ac.id

1. INTRODUCTION

In 2009 Kevin Ashton has introduced the term of the Internet of Things which consists of things and internet [1]. IoT refers to things which have the capability to sense, see and hear the environment and interconnected via a network to exchange data.

IoT is widely used in some areas such as Sabriansyah develops systems in smart homes [2], Rghioui describes the use of IoT for measuring human activities [3], Navulur describes the use of IoT in agriculture [4], and Gupta describes the use of IoT in monitoring child safety while going to school [5]. Those systems are developed through a variety of applications and devices types. However, the use of various devices on one platform to used gather and consume data from devices will have consequences.

Nowadays, we are dealing with various vertically oriented and mostly closed systems. IoT architectures are built on heterogeneous data protocols, e.g. Message Queuing Telemetry Transport (MQTT), Extensible Messaging and Presence Protocol (XMPP), Constrained Application Protocol (CoAP), and Lightweight M2M. Within such heterogeneous data protocols, development of simple applications on IoT requires skill and time that is not for a while, besides every new system development, developer is required to deal with the different basic functions. The most important task in IoT is providing interoperability to support object addressing between things [6].

According to Dinesh, interoperability in IoT falls into three layers namely Network Interoperability, Syntactical Interoperability, Semantic Interoperability. Network interoperability refers to information from things shared among different network and interface communication. Syntactical Interoperability refers to data protocol which used for exchange information or message, for example, CoAP, MQTT, XMPP, AMQP, DDS, and HTTP. From those protocols, only COAP and MQTT are more suitable for IoT, because both are designed for energy-efficient and low resources device. Whereas Semantic Interoperability refers to content and data models [7].

One of the solutions to overcome interoperability problems in IoT is developt a middleware that provide abstract layer between things with multiple communications protocols and the application by exposes universal API; therefore, developers can use it in their applications to consume data from things. Razzaque described one of middleware design that addresses interoperability issue in IoT is event-based middleware. This design also provides some advantages in reliability, availability, real-time performance, scalability and security [8].

Based on these, we propose an event-based middleware as intermediary multi-protocol gateway (a) to provide Syntactical interoperability between devices using CoAP and MQTT protocol, (b) extensible architecture so other developers can add another protocol they need, (c) unified API based on WebSocket to expose data from devices. This approach extends web oriented ability to perform real-time messaging and help developers to focus on their application rather than enabling technology on each device. The proposed middleware has been implemented and validated through an extensive performance evaluation.

The remainder of this paper is organized as follows: a discussion of previous work that has tried to address these problems in section 2, an overview of the middleware that we proposed in section 3, experiment setup in section 4, result about experiment and evaluation in section 5 and the con clussion in section 6.

2. RELATED WORKS

In this section, we present a related work to our IoT middleware; first, we analyze regarding the interoperability in the IoT; second, existing middleware addressed the interoperability problem; third, we focus on the literature regarding enabling technologies web-oriented in IoT; lastly, literature for the evaluation of our proposed middleware.

In [9] It is mentioned that interoperability is the biggest thread as the result of a huge variety of connected device that will interact, either existing one or to be discovered in the future. As stated by Desai [10], an IoT architecture must be self-contained and provide integrated and capable translation of various popular messaging protocols. Thomas Zachariah exposed a problem for IoT which is existing middleware (gateway and application) mostly can only be used by one particular type of device. For example, the smartwatch as Apple Watch, Fitbit, and Moto 360 requires certain applications as a gateway to connect to the internet. The same thing happened on the smart light bulb as Philips Hue that requires specific devices and applications to operate. To overcome this problem, researchers offer a gateway to the architectural concept smartphone based on Bluetooth Low Energy (BLE) and IPv6 [11]. Guinard introduced a semantic gateway which provides translation function between widely used MQTT, XMPP, and CoAP protocols and exposing physical device as a RESTful resource, but do not focus on the interactions of the gateway with external users [12].

Razzaque [8] classifies middleware's requirements into three. First, functional requirements which require a middleware to be used by a variety of devices with ease without having to pay attention to technology and data model used by the device; second, non-functional requirements including real-time, reliability, scalability, availability, security and privacy, and ease-of-deployment; Lastly, architectural requirements including adaptive and context-awareness, interoperability, autonomous behavior, and service-based. Our approach focuses on syntactical interoperability problem and enhances existing web-oriented by providing real-time communication.

For evaluation method, Reza has conducted research related to interoperability testing method. From several testing methods, interoperability assessment methodology is the only one that includes descriptive test (Yes / No) and qualitative testing [13]. Dinesh used several tests to evaluate the performance of COAP and MQTT via common middleware. He also examine Influence of packet loss on delay and data transfer, overhead for various message sizes and how middleware adaptively change its protocol based on the network condition. This method also measures the performance of the system such as message delivery and latency just like in [7]. Therefore, in this paper, we use this method to evaluate our middleware in terms of interoperability.

3. PROPOSED MIDDLEWARE ARCHITECTURE

The proposed middleware architecture is implemented using the "Ends to middle" model. There are three main functions: sensor gateway, service unit, and application gateway as illustrated in Figure 1. The architecture introduces an approach which implementing publish-subscribe pattern to connect CoAP and MQTT enabled devices then exposing their data through WebSocket API.

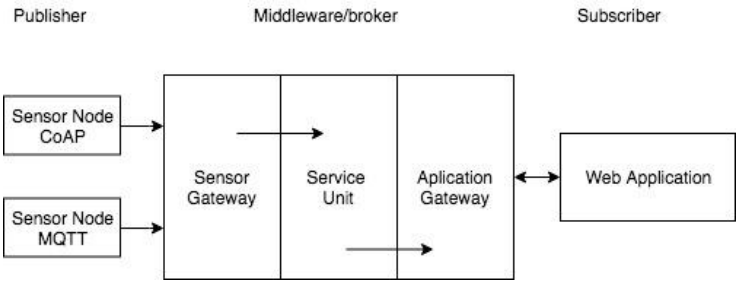

Figure 1. Middleware architecture

Further explanation of each component and how it works will be described as follows:

a. Sensor gateway

This component consists of two gateways; each is responsible for handling the incoming message from CoAP or MQTT enabled devices, respectively. As state-of-the-art of publish/subscribe pattern, each message should have a topic; MQTT has this topic by nature since it implements publish/subscribe too. On the other hand, we have to expose CoAP resource as topics. For example, if the topic's name is kitchen, so we can publish a message in the topic by doing a POST request at /topics/kitchen.

The complete process on how a gateway handles incoming message is shown in Fgure 2. After receiving a message from CoAP or MQTT device, it will check the integrity of URL and pass the data to Service Unit.

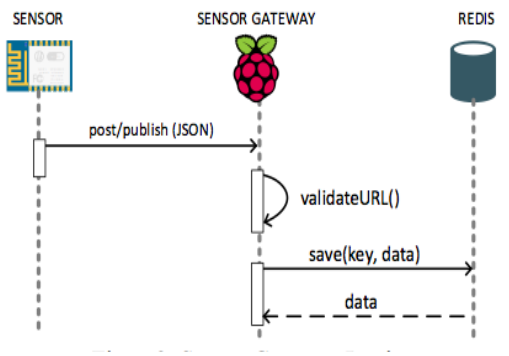

Figure 2. Sensor gateway logic

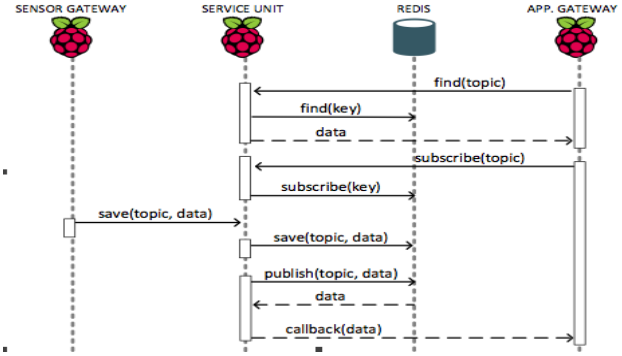

Figure 3. Service unit logic

b. Service unit

Service unit provides API's for Sensor and Application gateway to publish or subscribe a particular topic to Redis. This API's allow developers or users to add their own protocol apart from CoAP and MQTT. Redis has an important role in our system as a broker of the publish and subscribe mechanism as shown in Table 1.

Table 1. Service Unit's API

Interface	Description
find(topic)	
subscribe(key)	
save(key, payload)	

As shown in Figure 3, an application subscribes a particular topic through Application gateway. Under the hood, first, Service unit finds existing data for that topic and returns it directly if there is one. Alternatively, Service layer will make a subscription and set a listener to Redis using the topic as key. On the other side, when a device publishes message through Sensor gateway, Service unit will save and publish it to Redis. At the same time, every Service unit that set listener earlier get notified about new data and pass it through to Application gateway.

c. Application Gateway

This component basically exposes several APIs for applications to consume data from IoT devices using WebSocket protocol.

Table 2 shows the application gateway's API.

Table 2. Application Gateway's API

Interface	Description
connect()	
subscribe(topic)	

Given an application that needs to read data from IoT devices, it should connect to our middleware first so it can subscribe a topic. Every time an IoT device send data using the same topic to middleware, this application will receive the same data in real-time (Figure 4).

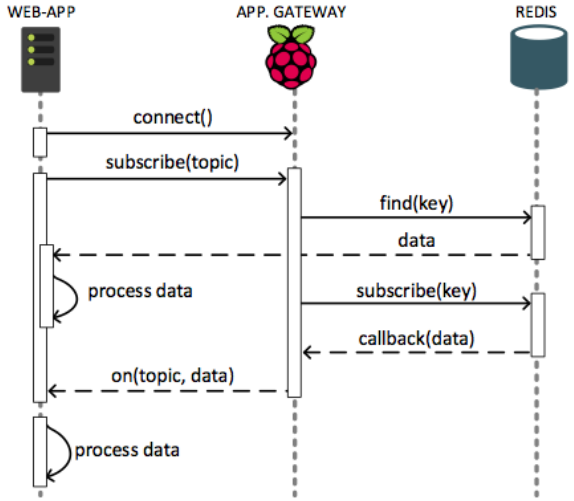

Figure 4. Application gateway logic

4. EXPERIMENT SETUP

In order to examine our middleware, we build a system consisting of several components which are temperature and humidity sensors, our middleware on Raspberry Pi, Web application, and MongoDB as data storage.

Our middleware connects IoT devices such as sensor and application, as shown in figure 5. Its main responsibility is to take incoming data from the sensor and notify subscriber of new topic, which is described as follows: The client connects to middleware through WebSocket and subscribes for update on a topic:

1. The web application registers to get an update on a specific topic, let say the topic is "/topics/kitchen".
2. There is a new event from a sensor node which delivers new data under topic "/topics/kitchen".
3. Whenever the sensor node publishes new data even through CoAP or MQTT protocol, it is forwarded to the Service unit;
4. Service unit stores the data and publishes new data to the subscriber with a specific topic.
5. After got updates, web apps display data to front-ends and store data on MongoDB.

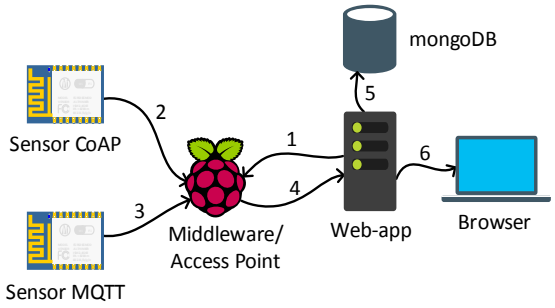

Figure 5. System architecture

4.1. Enabling technologies and implementation

a. Sensor devices

As shown in Figure 7, we use two identical sensors that implement CoAP and MQTT protocol. All sensors are built using NodeMCU (ESP8266) and DHT module. Next, we manage to set our sensors in order to send an update every 30 seconds. The data sent by sensor follows semantic as depicted in Figure 6.

Node MQTT

```
var payload = {
protocol : "mqtt",
        timestamp:new Date().toISOString(),
        topic : "home/mqtt1500",
        sensor : {
        tipe : "esp8266",
           index : "mqtt",
        ip : "192.168.42.245",
           module : "dht22"
        },
        humidity : {
           value : 20,
           unit : "%"
        },
        temperature : {
           value : 30,
           unit : "celcius"
        }
}
```

Node CoAP

```
var payload = {
protocol : "coap",
        timestamp : new Date().toISOString(),
        topic : "home/coap10",
        sensor : {
        tipe : "esp8266",
           index : "cocoap",
        ip : "192.168.42.245",
           module : "dht22"
        },
        humidity : {
           value : 20,
           unit : "%"
        },
        temperature : {
           value : 30,
           unit : "celcius"
        }
}
```

Figure 6. Semantic data

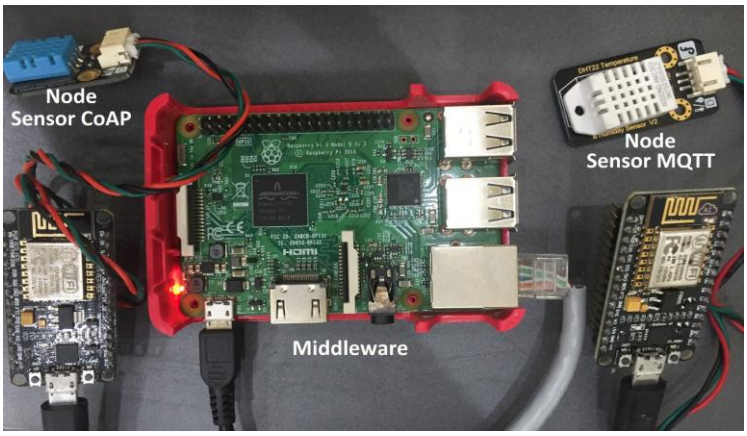

Figure 7. Node sensor and middleware hardware

b. Middleware

In particular, our middleware is developed on the top of the Node.js framework which has an event-driven paradigm.To provide communication using CoAP, MQTT and WebSocket protocol, we use the node-CoAP, MQTT.js, and Socket.io implementation, respectively. In the end, our middleware runs on a Raspberry Pi machine.

c. Web application

The function of the web application is to display the data provided by sensors in real time. Every time Web application get notified about new data, it will store it in MongoDB and update the web front-end.

d. NetworkTopology

Apart from running middleware, Raspberry Pi also acts as an Access Point to connect sensors and middleware. To achieve this, Raspberry Pi uses two network interfaces; eth0 connected to a campuslocal area network (LAN), and wlan0 set with static IP as the default gateway for access points provided. In order to communicate with middleware, all sensors should connect to Raspberry's Access Point. On the other hand, the subscriber connects to the LAN to consume data from sensor trough middleware. The complete topology used in this research shown in Figure 8.

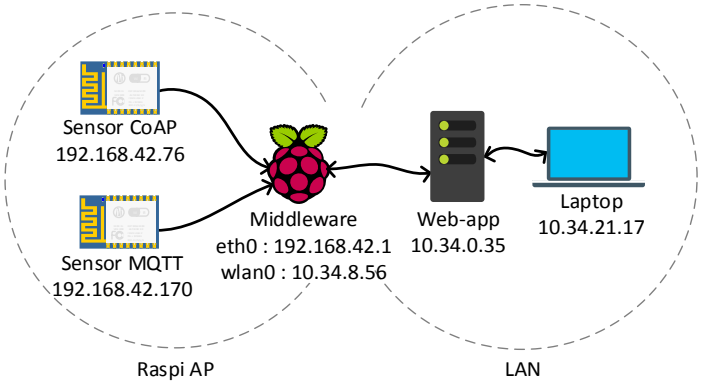

Figure 8. Network topology

5. EXPERIMENT AND EVALUATION RESULTS

Tomeasureperformanceofour middleware, we defineseveral parameters as shown in Table 3. Wireshark was used to measure data transferred from data sensor through the middleware to web server.

Table 3. Parameters for Evaluation

Parameter	Description	Result
Are system requirements common?	Ensure that all components in our system are already fulfilled, including temperature, and humidity sensor based on CoAP and MQTT, Raspberry Pi 2, Wireless adapter, Web serverand a laptop.	Y
Is network topology common?	Ensure that our network configuration is up and running based on topology that already defined. This covers the connection between sensors, Raspberry Pi, and Web server.	Y
Is there any semantic data used?	Set a standard semantic for all sensors to send data.	Y
Is data transmission satisfying??	We evaluate our middleware's performance by determining success rate on transmitting data according to this scenario: a. CoAP sensor to middleware and middleware to Web server b. MQTT sensor to middleware and middleware to Web server c. CoAP and MQTT sensor to middleware and middleware to Web server For each scenario, we monitor data transition for an hour. To get a better resultwe repeat this process three times each.	Y
Is data latency acceptable?	We measure the influence of packet loss to delays in the message reception and simulate various packet loss rate from 0%, 5%, 10%, 15% and 25% using network emulator [14] on Raspberry Pi access point. This measurement used the same scenario in success rate on transmitting data test.	Y

Parameter	Description	Result
Is semantic data received common?	We check data integrity from our system by comparing data sent by sensor and data saved in the Web server.	Y
Is correct action taken after receiving data?	We check that every data sent from a sensor is received by the Web server and saved in database successfully.	Y

The first thing that needs to be ensured is that the proposed system works well. As shown in the Figure 9, the middleware can receive data from sensor node using CoAP protocol with topic home/kitchen and data from sensor node using MQTT protocol with topic home/garage. At the same time, the middleware can handle request from subscriber to retrieve data for later saved in web server. Figure 10 shows example data from sensor node which is stored in web server with topic home/garage.

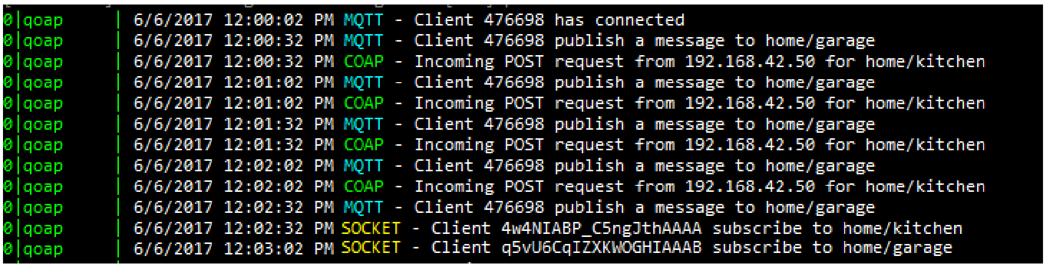

Figure 9. Monitoring activities in middleware

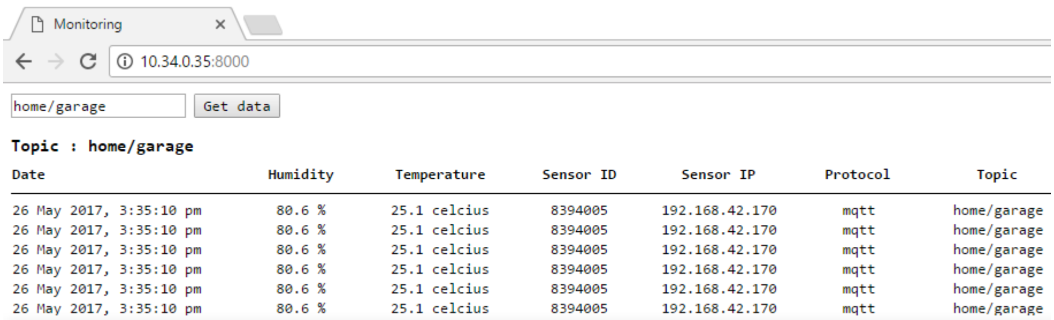

Figure 10. Web application

To answer data transmission satisfaction, we used two sensor nodes and run two delivery scenarios. In the first scenario, each sensor node simultaneously publishes data every 30 seconds using CoAP Non-Confirmable and MQTT QoS 0. In the second scenario sensor node publishes data using CoAP Confirmable and MQTT QoS 1. This experiment was conducted as long as four our, there are 120 data in each hour. We monitor and measure the success rate of both protocols when publishing data to middleware until it was received by the application.

This experiment results have shown that, if reliability feature is not activated, both protocol show variety in packet lost. This can be caused by the network environment. CoAP has average success rate of 92% with average packet loss 7.2%. Meanwhile, MQTT has average success rate of 90% with average packet loss 9.4%. This will impact the amount of data that should be available and taken by the subscriber. If reliability feature in both protocols is activated, we got 100% average success rate and 0 % packet loss. As shown in Table 4 and Table 5.

The next experiment is involving some packet loss conditions to answer data latency. We used network emulator to perform the condition of packet loss between 0 - 25%. The results obtained, due to packet loss, both CoAP and MQTT protocols will try to retransmit that packet. As the packet loss grows, the number of retransmission.

Table 4. Success Rate of CoAP and MQTT Protocol Transmission

CoAP				MQTT				Websocket			
Expected	Actual	Success rate	Packet loss rate	Expected	Actual	Success ate	Packet loss rate	Expected	Actual	Success rate	Packet loss rate
120	112	93.33%	6.67%	120	117	97.50%	2.50%	240	231	96.25%	3.75%
120	118	98.33%	1.67%	120	111	92.50%	7.50%	240	230	95.83%	4.17%
120	104	86.67%	13.33%	120	98	81.67%	18.33%	240	202	84.17%	15.83%
120	111	92.78%	7.22%	120	109	90.56%	9.44%	240	221	92.08%	7.92%

Table 5. Success Rate of CoAP Confirmable and MQTT QoS 1

CoAP				MQTT				Websocket			
Expected	Actual	Success rate	Packet loss rate	Expected	Actual	Success ate	Packet loss rate	Expected	Actual	Success rate	Packet loss rate
120	120	100%	0%	120	120	100%	0%	240	240	100%	0%
120	120	100%	0%	120	120	100%	0%	240	240	100%	0%
120	120	100%	0%	120	120	100%	0%	240	240	100%	0%
120	120	100%	0%	120	120	100%	0%	240	240	100%	0%

Table 6. Average delay for different packet loss

Packet Loss	Average Delay			
	CoAP SN→ middleware	MQTT SN → middleware	Ws Middleware → subscriber	Ws Middleware → subscriber
0%	0.006944	0.003954	0.008366	0.005377
5%	0.210761	0.119539	0.212129	0.120907
10%	0.373347	0.342412	0.374707	0.343772
15%	0.624716	0.463573	0.626127	0.464984
20%	0.831597	0.661714	0.832958	0.663075
25%	1.068280	0.770427	1.070014	0.772161

Table 6 shows the average effect of delay on publishing data from sensor nodes (SN) to middleware using CoAP and MQTT. It also affects the delay of WebSocket. From the experimental results, obtained packets sent using MQTT experience smaller number of packet loss, but Figure 11 and Figure 12 conclude that CoAP works better than MQTT when higher packet loss occurs. This is because TCP overhead goes up, in contrast to the smaller UDP overhead.

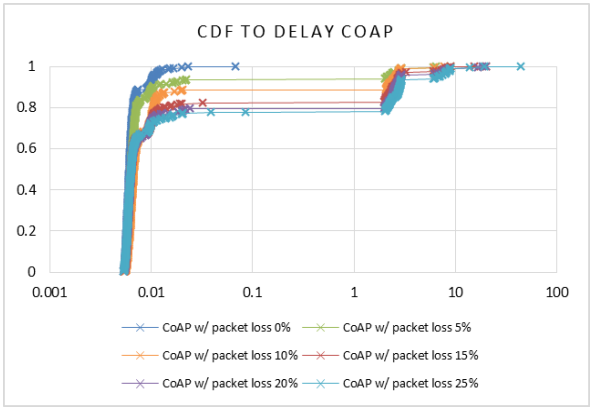

Figure10. Influence of packet loss rates CoAP on delay CDFs

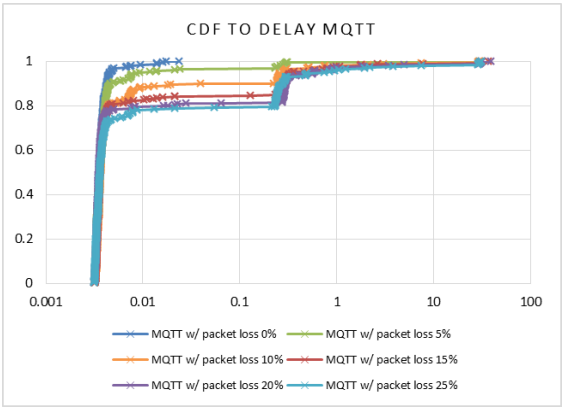

Figure11. Influence of packet loss rates MQTT on delay CDFs

REFERENCES

[1] K. Ashton, "That 'Internet of Things' Thing, in the Real world things matter more than ideas", *RFID J.*, 2009.
[2] S. R. Akbar, *et al*., "Design of Pervasive Discovery, Service and Control for Smart Home Appliances: An Integration of Raspberry Pi, UPnP Protocols and Xbee", *International Journal of Electrical and Computer Engineering*, vol. 7, no. 2, pp. 1012-1022, 2007.
[3] A. Rghioui and A. Oumnad, "Internet of Things: Surveys for measuring human activities from everywhere," *International Journal of Electrical and Computer Engineering*, vol. 7, no. 5,

pp. 2474-2482, 2017.
[4] S. Navulur, C. S. S. AS, and G. P. MN, "Agricultural Management through Wireless Sensors and Internet of Things," *International Journal of Electrical and Computer Engineering*, vol. Vol 7, no. 6, pp. 3492-3499, 2017.
[5] P. Gupta, *et al*., "An IoT Framework for addressing parents concerns about safety of school going children," *International Journal of Electrical and Computer Engineering*, vol. 6, no. 6, pp. 3052-3059, 2016.
[6] P. Barnaghi, *et al*., "Semantics for the Internet of Things," *Int. J. Semant. Web Inf. Syst.*, vol. 8, no. 1, pp. 1-21, 2012.
[7] D. Thangavel, *et al*., "Performance Evaluation of MQTT and CoAP via a Common Middleware," in *IEEE ISSNIP 2014 - 2014 IEEE 9th International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Conference Proceedings*, 2014.
[8] M. A. Razzaque, *et al*., "Middleware for Internet of Things: A survey," *IEEE Internet Things J.*, vol. 3, no. 1, pp. 70-95, 2016.
[9] T. Itala and M. Hamalainen, "Gateway as a Service: A Cloud Computing Framework for Web of Things," *2012 19th Int. Conf. Telecommun.*, no. Ict, pp. 1-6, 2012.
[10] P. Desai, *et al*., "Semantic Gateway as a Service Architecture for IoT Interoperability," in *Proceedings - 2015 IEEE 3rd International Conference on Mobile Services, MS 2015*, 2015, pp. 313-319.
[11] T. Zachariah, *et al*., "The Internet of Things Has a Gateway Problem," *Proc. 16th Int. Work. Mob. Comput. Syst. Appl. (HotMobile '15)*, pp. 27-32, 2015.
[12] D. Guinard, *et al*., "A Resource Oriented Architecture for the Web of Things," *Proc. 2010 Internet Things*, pp. 1-8, 2010.
[13] R. Rezaei, *et al*., "Interoperability Evaluation Models: A Systematic Review," *Comput. Ind.*, vol. 65, no. 1, pp. 1-23, 2014.
[14] The Linux foundation, "Networking:netem," 2016. [Online]. Available: https://wiki.linuxfoundation.org/networking/netem. [Accessed: 12-Feb-2017].