WoTPubSub: A Web of Things Middleware with Publish Subscribe Functionality for Enabling Lightweight and Efficient Standard Web Access on Constrained Device

Adhitya Bhawiyuga, Donny Kurniawan, Reza Andria Siregar, Widhi Yahya, Dany Primanita Kartikasari

Abstract—The web platform can be seen as an auspicious candidate to provide an interoperability layer in an IoT based system with various kind of device specification and client platform leading to the transformation from IoT to WoT (Web of Things). In order to implement web platform on IoT world, we require a web compatible middleware yet still maintaining lightweight and efficient machine-to-machine (M2M) communications. In this paper we propose the web of things (WoT) middleware with publish subscribe functionality or WoTPubSub. As opposed to the existing solution, this middleware offers the utilization of lightweight MQTT protocol to perform a communication with constrained device while still maintaining the compatibility with existing web architecture. The proposed system consists of three actors: the user as Restful HTTP client, the sensing-actuating constrained device as both MQTT publisher-subscriber and the proposed middleware acting as communication bridge which translates user's HTTP request into MQTT publish-subscribe action. We consider two data flow scenarios in the proposed middleware: user obtaining data from sensing device and user giving a command to actuating device. From functional and performance testing, we conclude that the proposed middleware has been able to provide a web compatible intermediary functionality between user and sensing-actuating constrained device with improved performance compared to the existing approaches.

Keywords: web of things; publish subscribe; constrained device

I. INTRODUCTION

The advancement on embedded and networking technology have contributed to the increasing number of physically connected devices. It leads to the realization of Internet of Things (IoT) concept in which every physical device can be accessed from anywhere through the internet.

Revised Manuscript Received on October 30, 2019.

* Correspondence Author
Adhitya Bhawiyuga*, Faculty of Computer Science, University of Brawijaya, Malang, Republic of Indonesia. (E-mail: bhawiyuga@ub.ac.id)
Donny Kurniawan, Faculty of Computer Science, University of Brawijaya, Malang, Republic of Indonesia
Reza Andria Siregar, Faculty of Computer Science, University of Brawijaya, Malang, Republic of Indonesia
Widhi Yahya, Faculty of Computer Science, University of Brawijaya, Malang, Republic of Indonesia.
Dany Primanita Kartikasari, Faculty of Computer Science, University of Brawijaya, Malang, Republic of Indonesia

In order to accommodate that requirement, various communication protocols have been invented by researchers such as IEEE 802.11, IEEE 802.15.4, Bluetooth Low Energy (BLE), 6LowPan and so on [1]. While those different protocols perform well in the local wireless sensor network (WSN) scenario, it may still require an interoperability layer so that those various devices can be represented in an unified way [2].

The web platform can be seen as an auspicious candidate to provide such interoperability layer which then implicates to the conceptual transformation from internet to the Web of Things (WoT) [4]. As an advantage of WoT concept, every device, regardless of its hardware and communication platform, can be abstracted in a single way i.e. as web resource [5]. It then opens the possibility to integrate the IoT devices with another application such as in smart home monitoring-controlling system. In addition, a user can flexibly access those devices with various kinds of client platforms such as web and mobile application.

In order to provide an interface to a WoT based system, we require a networking protocol that complies with web standard. The Hypertext Transfer Protocol (HTTP) has been widely used in web technology as well as in machine-tomachine (M2M) communication through the use of webservice [6]. HTTP utilizes the half-duplex request-response stateless model with two main actors : HTTP client and server. Any HTTP request is marked with two main part in its header : HTTP method to distinguish the action and unified resource locator (URL) to mark the server resource location. As an addition to HTTP, people invent an architectural pattern called Representational State Transfer (REST) to better support the M2M communication [7]. In this sense, the REST is actually only a set of specifications, including method and URL, to access a specific resources and functionalities using standard HTTP request-response messaging. In literature, there exist two main methods to implement RESTful HTTP protocol on IoT world namely : device-as-server and middleware-as-server approaches. In the first approach, researchers develop a micro native HTTP server and deploy it into sensor equipped device [8], [9], [10]. In this case, the device is acting as a server while the user or other applications are in client side demanding the device's resource and service through standard HTTP request.

On one side, this approach offers a simple WoT design and architecture to the realm. However, since the device is constrained in term of computation and energy, the direct HTTP deployment can possibly drain those device resources. In the second approach, a middleware is introduced as an intermediary between the user and device [11], [12], [13], [14]. In this case, the HTTP server is deployed on middleware while both user and device are acting as HTTP client. The middleware periodically receives the sensor data from device and stores it for further access by user. On the other direction, for giving a specific order to actuator device, the user sends out a message to middleware. The device then periodically check for that incoming command by sending out the HTTP request to middleware. Although the second approach moves the burden of being HTTP server from device to middleware, however, an additional overhead arises as the device still need to periodically emit a request to middleware to get the latest command from user. This condition can be severe as the HTTP is required to perform a new TCP connection establishment every time a client request is emitted. Therefore, a data oriented networking protocol with less overhead is required to facilitate the communication between the middleware and constrained device.

One of possible candidate to cope with this challenging issue is Message Queue Telemetry Transport (MQTT) protocol. In contrast with HTTP, MQTT is a lightweight TCP based messaging protocol with indirect communication paradigm in mind [17]. It consists of three main actors : publisher, subscriber and broker. Publisher has a role of sending the data labeled by a specific topic to broker [3]. The broker then relays the message to all subscribers who are interesting on that particular topic. Therefore, instead of identified by certain sender or receiver address, a message is identified by a specific label agreed upon sender and receiver. This may leads to an efficient communication pattern since if a publisher would like to send a similar message to certain group of subscribers, it only needs to emit one message with a specific topic and let the broker to relay that message to all subscribers with similar interest [15], [16]. In addition, a MQTT client, either publisher or subscriber, is only required to perform a TCP handshaking at once during first connection establishment which is then maintained by broker using a small-sized heartbeat message until one of communicating entity breaks it. This lightweight and efficient communication pattern is more suitable for constrained device case since it can possibly reduce the communication overhead while still maintaining its efficiency.

This paper proposes the web of things (WoT) middleware with publish subscribe functionality or WoTPubSub for enabling web compatible yet lightweight and efficient machine-to-machine (M2M) communications. As opposed to the existing solution, this middleware offers the utilization of lightweight MQTT protocol to perform a communication with constrained device while still maintaining the compatibility with existing web architecture. The proposed system consists of three actors : the user as Restful HTTP client, the sensing-actuating constrained device as both MQTT publisher-subscriber and the proposed middleware acting as communication bridge which translates user's HTTP request into MQTT publish-subscribe action. There exists two scenarios considered in the proposed system : user obtaining data from sensing device and user giving a command to actuating device. At first scenario, the device periodically publishes its sensing data to the subscriber deployed middleware in which the data will be temporarily stored for further access. The user can then obtain that latest data by emitting a specific HTTP request and receives the data as its response. On the reverse direction i.e. second scenario, the user first sends a specific command wrapped on a HTTP request to middleware which then translates it into a MQTT publish message to the device acting as subscriber. The remainder of this paper is organized as follows. We first present our requirement analysis in section II. Then, in sections III and IV, we describe the design and implementation of the proposed system. Then, we discuss our functional and performance testing results in section V. Finally, we conclude our paper at section VI.

II. REQUIREMENT ANALYSIS

Based on the problem stated in section I, the proposed system should provide following features :

1) The sensing device should be able to publish the sensor data with a specific topic.

2) The middleware should provide subscriber component to receive sensor data from sensing device and store it for further access.

3) The middleware should provide a Restful HTTP server instance with two basic functionalities :

a. receiving user command and then publish it to actuating device

b. receiving sensor data request from user and response it with stored sensing data.

4) The user should be able to send a standard HTTP request and receive its response accordingly.

III. SYSTEM DESIGN

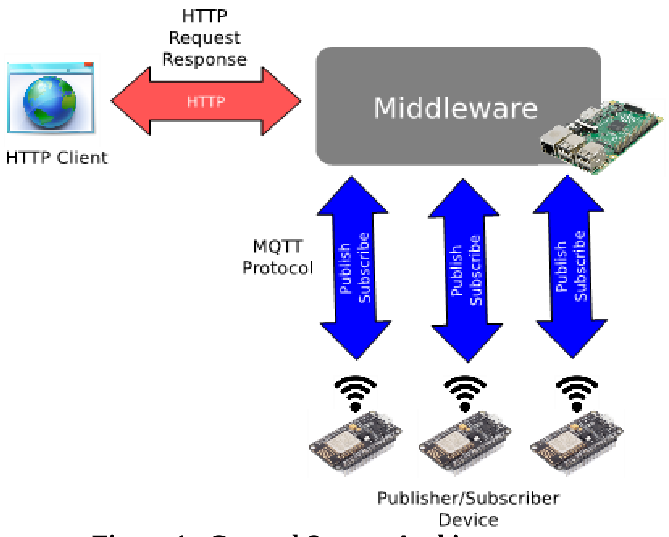


Figure 1. General System Architecture

In this section we present the system design and the scenario considered in proposed middleware. Fig. 1 illustrates the general architecture of the system with three main actors :

1) the user as Restful HTTP client
2) the uniquely identified sensing-actuating device as both MQTT publisher-subscriber
3) the proposed middleware acting as communication bridge which translates HTTP request from user into MQTT publish-subscribe action.

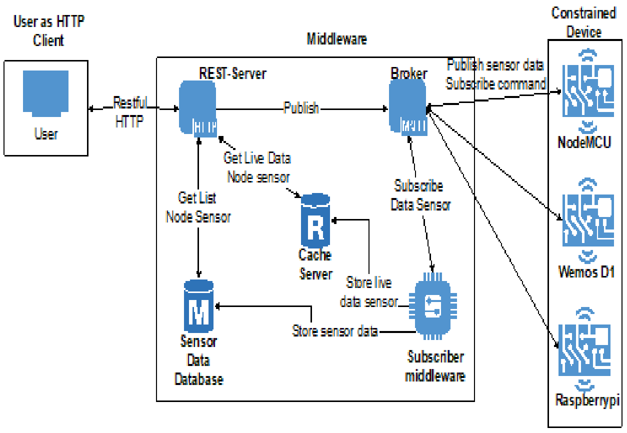
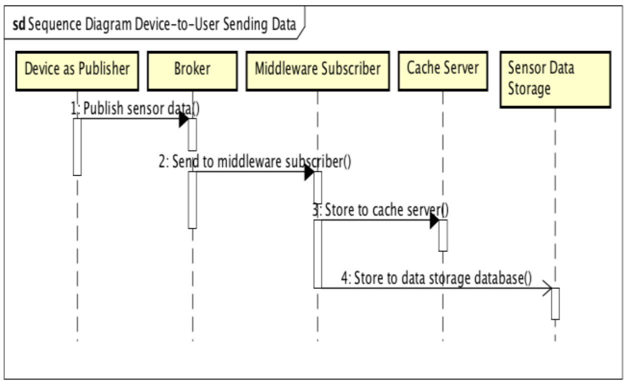


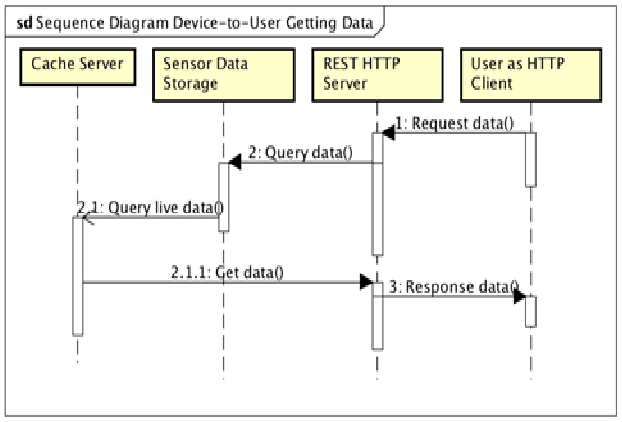
Figure 2. Detail Sub Component of Middleware.

As presented in Fig. 2, the middleware part can then be decomposed into more detailed sub component including the Restful HTTP server to receive any request from user, MQTT broker as messaging server for both sensing-actuating device, and generic sensor data subscriber to receive and store data obtained indirectly from sensing device. In general, there exists two main scenario considered in proposed middleware : user obtaining data from sensing device and user giving a command to actuating device.

3.1. Device-to-User Scenario


sd Sequence Diagram Device-to-User Sending Data
Device as Publisher
Broker
Middleware Subscriber
Cache Server
Sensor Data Storage
1: Publish sensor data()
2: Send to middleware subscriber()
3: Store to cache server()
4: Store to data storage database()

a) Sending Sensor Data from Device to Middleware



a) Getting Sensor Data from Middleware

Figure 1. Sequence Diagram of Device-to-User Scenario

The first scenario is considered to provide an user access to the stored sensor data. Fig. 3 presents the sequence diagram of to illustrates the device-to-user scenario. At first, the device periodically publishes its sensing data to the generic subscriber deployed in the middleware part. In this case, to identify a sensing data from different device, each published message is labeled with topic containing the device unique identifier. For example, the sensing data from device ”1” and ”2” can be labeled with topic ”/sensor/1” and ”/sensor/2”, respectively. The generic subscriber can then receive the data coming from all device by simply subscribing to a wildcarded topic e.x. ”/sensor/#”.

Upon reception, the generic subscriber then store the data to a local database for persistence storage. In order to improve the data access performance, the subscriber temporarily stores the latest sensing data to a memory caching service. When the user sends a request to obtain sensing data, the Restful HTTP server performs a query from either memory caching service for latest data or from local database for historical data access. The data obtained from query process are then emitted back to the user as standard HTTP response. For example, if user would like to get sensor data from sensing device ”1” containing temperature and humidity information, he/she can utilize HTTP request message as presented in Table I.

Table I. Request and Response Format for User-to-Device Scenario

Request : GET /device/1 HTTP/1.1
Response HTTP/1.1 200 OK *{”temperature” : 30, ”humidity” : 80}*

3.2. Device-to-User Scenario

On reverse direction of the first scenario, the user can possibly sends out a command request to an actuating device as illustrated in Fig. 4 . To do so, the user first emits a command containing request with specific method and Unified Resource Locater (URL) to Restful HTTP server deployed on middleware. Notice that, the URL is utilized to identify each ifferent device.

Once that request received, the server then performs the translation of URL into MQTT topic which is then utilized to label the published command messsage. In this case, each actuating device subscribes different unique topic while the middleware is acting as command publisher.

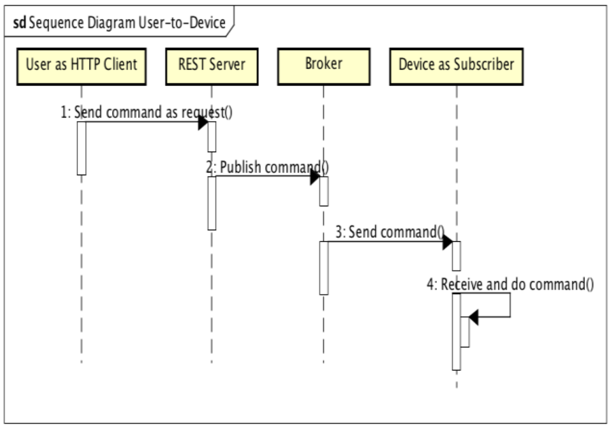


Figure 2: Sequence Diagram of User-to-Device Scenario.

For example, if user would like to turn on the lamp con trolled by actuating device ”1”, he/she can send the message request as presented in Table 2.

Table 2. Request and Response Format for User-to-Device Scenario.

Request : POST /device/1 HTTP/1.1 {”act”:”on”, ”dev”:”lamp” }
Response HTTP/1.1 200 OK {”results”:”Action has been sent.”}

IV. IMPLEMENTATION

Based on the design presented in Section III, we then implement each of middleware sub component in term of its hardware and software.

4.1. Sensing-Actuating Constrained Device as Publisher/Subscriber

In this research, we utilize NodeMCU system on chip board as main controller of sensing-actuating device with following specification :

- ESP-8266EX microcontroller integrated with IEEE 802.11 b/g/n communication module
- 80 Mhz clock speed
- 128KB memory
- 4MB storage.

The board is attached with DHT-11 sensor to obtain the temperature and humidity of surrounding environment. Then, to perform an actuating action, we connect the main board with the LED lamp.

We then develop a program using Arduino framework which is then deployed on the main controller. Listing 1 presents the pseudocode of software part. The program first connect to the MQTT broker with specific IP address and port. Once the connection established, the device waiting for incoming user command by subscribing to a specific topic as defined in Section III. In this case, every incoming message relayed from broker will be handled in function message callback. In the other side, the device also periodically publishes its sensing data to the broker represented in loop function.

Listing 1. Pseudocode of Sensing-Actuating Device

```
message_callback(command){
 if(command["act"] == "on")
setDigitalPin(PIN, HIGH)
 else
setDigitalPin(PIN, LOW)
}
setup(){
mqtt_client.begin(broker_ip, broker_port)
mqtt_client.connect()
mqtt_client.subscribe("/device/device_id/
actuator")
mqtt_client.on_message = message_callback
}
loop(){
sensor_data = read_sensor()
mqtt_client.publish("/device/device_id/sensor",
sensor_data)
}
```

4.2. Middleware Device

We deploy all middleware components into a Raspberry Pi 3 minicomputer with following specifications :

- Broadcom BCM2837 SoC
- 1.2 GHz quad-core ARM Cortex A53 • 1GBRAM
- 2.4GHz 802.11n wireless communication module

4.3. Restful HTTP Server

In this paper, the middleware's Restful HTTP server is implemented using Python based web framework called Flask. The specification of application programming interface (API) in term of the HTTP method and its URL is presented in Table 3.

Table 3 . Request Format for Sensor Data Access

Request	Description
GET /devices	Get all available device with its identifier
GET /device/device id/sensor	Get latest sensor data from a specific device
GET /device/device id/sensor/all	Get all sensor data from a specific device
POST /device/device id/actuator	Send command to a specific device

4.4. Restful HTTP Server

We implement general subscriber on middleware using Python based MQTT client library called Paho MQTT. Listing 2 presents pseudocode of middlware subscriber. We first connect to the broker with specific IP address and port. Once connected, we subscribe to widlcarded topic ”/device/#” to receive sensor data from all devices with unique topic.

If a data is received, the program will invoke a callback function *message callback* and store sensor data to the storage.

Listing 2 : Pseudocode of Middleware Subscribe

```
message_callback(topic, data):
store_to_mongodb(data)
cache_to_redis(topic, data)
mqtt_client.begin(broker_ip, broker_port)
mqtt_client.connect()
mqtt_client.subscribe("/device/#")
mqtt_client.on_message = message_callback
```

In this paper we utilize a non-relational database called MongoDB to store persistence sensor data for further access [18]. This kind of database system is utilized since it offers a schema-less feature which leads to a flexible changes of data structure in future. This feature is important since the variety of sensors in a device can be extended in the future. Furthermore, to enhance the speed of latest sensor data access, we make use of temporary key-value cache system called Redis with topic as key and sensor data as value [19].

V. RESULT AND ANALYSIS

In this section, we present the result and analysis of performance testing on proposed middleware system. We utilize JMeter to emulate HTTP requests from 50, 100, 150 concurrent users. We then compare the performance of proposed middleware with two existing web of things implementation methods as stated in Section I namely device-as-server and middleware-as-server approaches.

5.1. Performance Comparison with Device-as-Server Approach

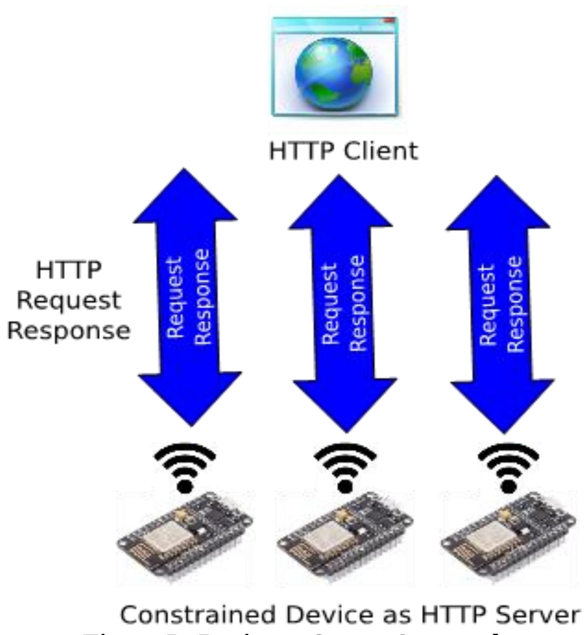
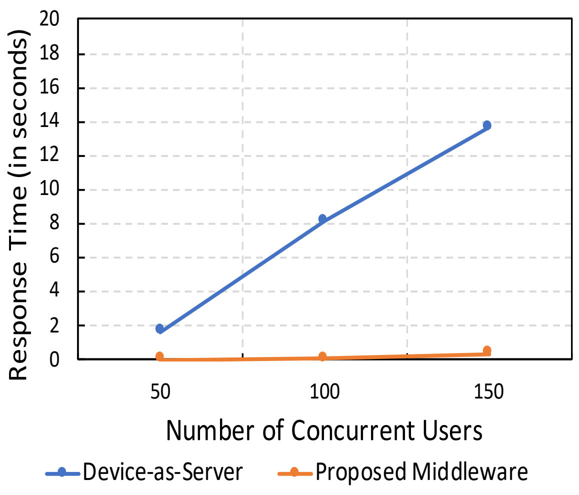


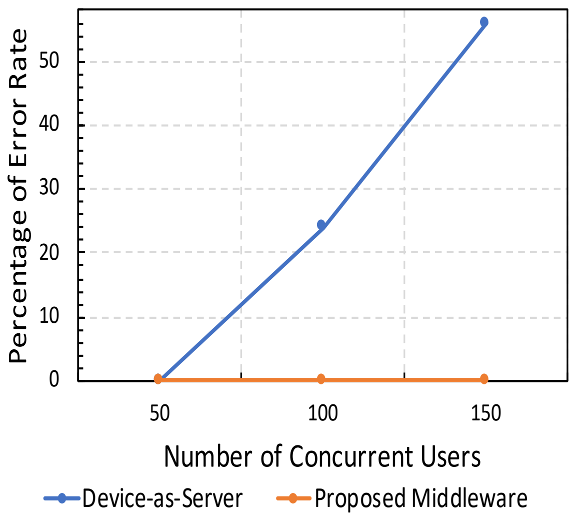
Figure 5. Device as Server Approach

Figure 5 illustrates the existing device-as-server approach tobe considered in this testing as opposed with the proposed middleware approach presented in Section III. In device-as-server case, we develop a tiny native webserver to handle any HTTP request from user and then deploy it to NodeMCU based constrained device. By using JMeter, we emulate HTTP client request from various number of concurrent users and then measure two parameters : response time expressed in seconds and server error rate representing the ratio of unanswered HTTP client requests.

The result of this testing is presented in Figure 6 (a) and (b) for response time and server error rate, respectively. From the result we can conclude that the response time and error rate of existing device-as-server approach increases as the number of concurrent users. This can be happened since the computational burdens for maintaining connection and giving the response to client request is heavier in constrained device side. In contrast with that existing approach, the proposed method shows a relatively good performance increase since the heavy computational burden is moved to middleware device with higher processing and memory capabilities.



(a) Response Time



(b) Server Error Rate

Figure 3 . Result of Performance Comparison with Device-as-Server Approach

5.2. Performance Comparison with Middleware-as-Server Approach

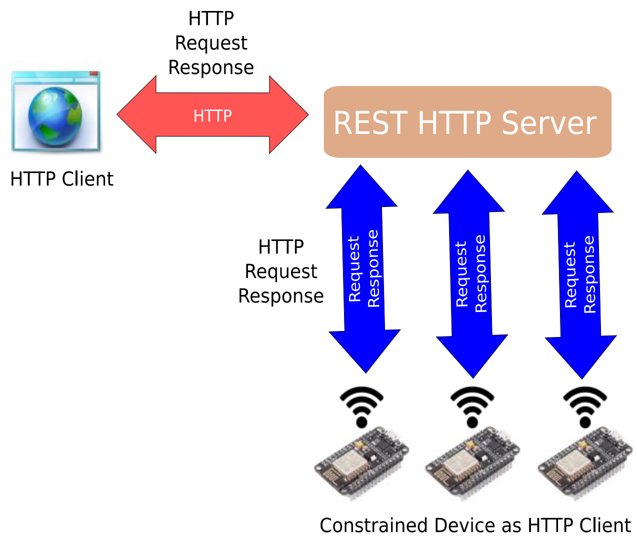
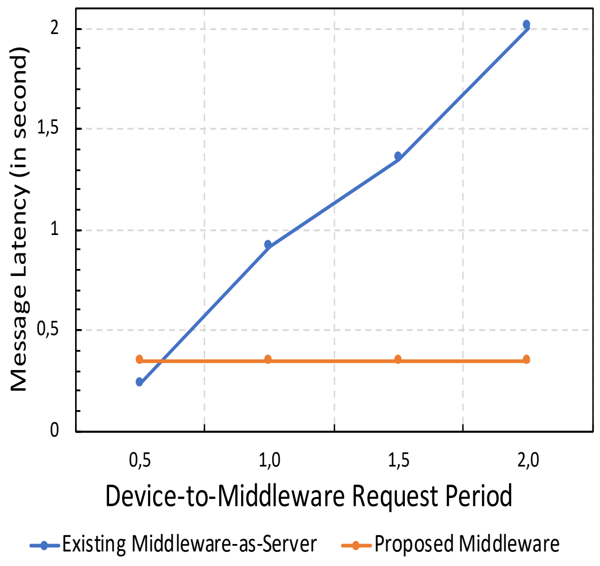


Figure 4. Existing Middleware-as-Server Approach.

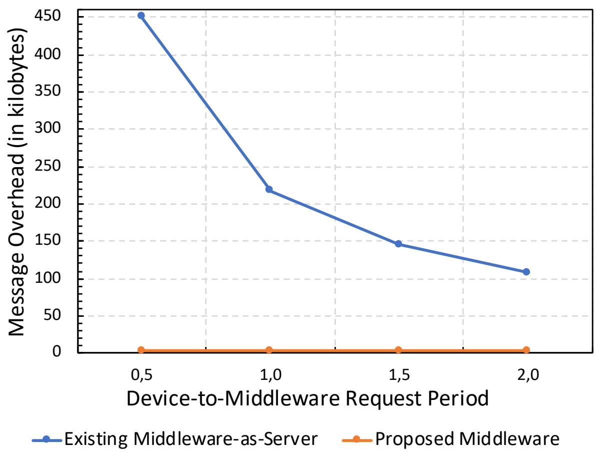
In this scenario we perform a performance testing to compare the existing middleware-as-server solution with proposed middleware. Figure 7 presents the architecture of existing middleware-as-server considered in this testing as compared with proposed one presented in Section III. The existing approach shares similarity with the proposed middleware as they deploy the HTTP webserver to middleware instead of in constrained device. However, as opposed with existing solution, the proposed middleware offers the utilization of lightweight MQTT protocol to perform a communication with constrained device.

This testing is conducted by sending a HTTP request command from user to the device through middleware intermediary. We then measure user-to-device message latency as well as its efficiency in term of message overhead in both existing and proposed schemes. Notice that, the overhead is calculated from all message required by a device to get command from middleware including request-response message of HTTP and periodic heartbeat packet in MQTT.

Figure 8 (a) and (b) presents the user-to-device message latency and its accumulative overhead during 5 minutes packet traffic observation for different device-to-middleware request intervals i.e. 0.5, 1, 1.5 and 2 seconds. From the figure, we can conclude that there exists a conflicting factor between the request interval and its required overhead in existing solution while the proposed middleware shows a steady overhead trend. If the request interval is lower, then a command can be received by device with lower latency. However, it may inquire additional cost on message overhead since the HTTP makes use of request-response architecture. Therefore, a client need to periodically send request to get the command as its response.



(a) Response Time



(b) Message Overhead

Figure 5. Result of Performance Comparison with Middelware-as-Server Approach

VI. CONCLUSION

We proposed the web of things (WoT) middleware with publish subscribe functionality or WoTPubSub. The proposed system consists of three actors : the user as Restful HTTP client, the sensing-actuating constrained device as both MQTT publisher-subscriber and the proposed middleware acting as communication bridge which translates user's HTTP request into MQTT publish-subscribe action. We consider two data flow scenarios : user obtaining data from sensing device and user giving a command to actuating device. From functional and performance testing, we concluded that the proposed middleware has been able to provide a web compatible intermediary functionality between user and sensing-actuating constrained device with improved performance compared to the existing solution.

VII. ACKNOWLEDGMENT

This work has been funded by Information Centric Networking Research Group (ICN-RG), Faculty of Computer Science, University of Brawijaya.

REFERENCES

1. Atzori, L., Iera, A., &Morabito, G. (2010). The Internet of Things: A survey. Computer Networks, 54(15), 27872805.
2. Al-fuqaha, A., Member, S., Guizani, M., Mohammadi, M., & Member, S. (2015). Internet of Things : A Survey on Enabling, 17(4), 23472376. [3]
3. Banks, Andrew, and Rahul Gupta. ”MQTT Version 3.1. 1.” OASIS standard 29 (2014).
4. Heuer, J., Hund, J., & Pfaff, O. *Toward the web of things: Applying web technologies to the physical world*. Computer, 48(5), 3442, 2015.
5. Raggett, D. *The Web of Things: Challenges and Opportunities*. Computer, 48(5), 2632, 2015.
6. Colitti, Walter, Kris Steenhaut, and Niccol De Caro. *Integrating wireless sensor networks with the web*. Extending the Internet to Low power and Lossy Networks (IP+ SN 2011) (2011).
7. Richardson, Leonard, and Sam Ruby. RESTful web services. ” O’Reilly Media, Inc.”, 2008.
8. W. Drytkiewicz, I. Radusch, S. Arbanowski, and R. Popescu-Zeletin, *pREST: a REST-based protocol for pervasive systems*,” in Proc. MAHSS, 2004, pp. 340348.
9. T. Luckenbach, P. Gober, S. Arbanowski, A. Kotsopoulos, and K. Kim, ”*TinyREST: A protocol for integrating sensor networks into the internet*,” in Proc. REALWSN, Stockholm, Sweden, 2005.
10. V. Gupta, A. Poursohi, and P. Udupi, *”Sensor.Network: An open data exchange for the web of things,*” in PERCOM Workshops, Mannheim, 2010, pp. 753755
11. Gao, Lei, Chunhong Zhang, and Li Sun. *”RESTful Web of Things API in sharing sensor data.”* Internet Technology and Applications (iTAP), 2011 International Conference on. IEEE, 2011
12. Tseng, Chih-Lung, and Fuchun Joseph Lin. ”Extending scalability of IoT/M2M platforms with Fog computing.” Internet of Things (WF-IoT), 2018 IEEE 4th World Forum on. IEEE, 2018.
13. Elmangoush, Asma, et al. ”Design of RESTful APIs for M2M services.” Intelligence in Next Generation Networks (ICIN), 2012 16th International Conference on. IEEE, 2012.
14. Li,Hongkun,et.al.”EnablingsemanticsinanM2M/IoTservicedelivery platform.” Semantic Computing (ICSC), 2016 IEEE Tenth International Conference on. IEEE, 2016.
15. P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, The Many Faces of Publish/Subscribe, ACM Comput Surv, vol. 35, no. 2, pp. 114131, Jun. 2003.
16. A. Sahadevan, D. Mathew, J. Mookathana, and B. A. Jose, An Offline Online Strategy for IoT Using MQTT, in 2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud), 2017, pp. 369373.
17. Bhawiyuga, Adhitya, Dany Primanita Kartikasari, and Eko Sakti Pramukantoro. ”A publish subscribe based middleware for enabling real time web access on constrained device.” Information Technology and Electrical Engineering (ICITEE), 2017 9th International Conference on. IEEE, 2017.
18. MongoDB, C. R. U. D. ”Introduction-MongoDB Manual 3.4.” (2017).
19. Carlson, Josiah L. ”Redis in action”. Manning Publications Co., 2013.