TELKOMNIKA Telecommunication, Computing, Electronics and Control
Vol. 19, No. 5, October 2021, pp. 1544~1552
ISSN: 1693-6930, accredited First Grade by Kemenristekdikti, Decree No: 21/E/KPT/2018
DOI: 10.12928/TELKOMNIKA.v19i5.20412

Enabling seamless communication over several IoT messaging protocols in OpenFlow network

Fauzi Dwi Setiawan Sumadi[1], Agus Eko Minarno[2], Lailis Syafa'ah[3], Muhammad Irfan[4]
[1,2]Department of Informatics, Universitas Muhammadiyah Malang, Malang, Indonesia
[3,4]Department of Electrical Engineering, Universitas Muhammadiyah Malang, Malang, Indonesia

Article Info

Article history:

Received Jul 16, 2020
Revised Jul 14, 2021
Accepted Jul 25, 2021

Keywords:

CoAP
Conversion server
IoT
MQTT
OpenFlow

ABSTRACT

The most prominent protocols for data transfer in internet of things (IoT) are message queuing telemetry transport (MQTT) and constrained application protocol (CoAP). The existing clients from both sides are unable to communicate directly because of the packet's header structure difference in application and transport layer. In response, this paper aims to develop a bidirectional conversion server used to translate the specified messaging protocol interchangeably in the OpenFlow network and transmit the converted packet from both sides. The conversion server integrated the MQTT subscriber and CoAP POST object for converting the MQTT message into CoAP data. Similarly, the CoAP-MQTT translation was processed by CoAP GET and MQTT publisher object. The research was evaluated by analysing the round trip time (RTT) value, conversion delay, and power consumption. The RTT value for MQTT-CoAP required 0.5 s while the CoAP-MQTT was accumulated in 0.1 s for single-packet transmission. In addition, the SDN controller and the conversion server only consumed less than 1% central processing unit (CPU) usage during the experiment. The result indicated that the proposed conversion server could handle the translation even though there was an overwhelming request from the clients.

Corresponding Author:

Fauzi Dwi Setiawan Sumadi
Department of Informatics
Universitas Muhammadiyah Malang
246 Raya Tlogomas Street, Malang 65144, East Java, Indonesia
Email: fauzisumadi@umm.ac.id

1. INTRODUCTION

IoT emerges as advancement in networking technology which utilises the unsophisticated devices to be able to directly connected to the global network. According to the basic concept, there exists a node called things which could perform a specific function such as delivering a sensing data obtained from its environment, actuating action based on specific conditional pattern, or even predicting the possible outcomes from the gathered data [1]. The extension of internet of things (IoT) in our daily life creates a systemic function that can be considered as a smart ecosystem where human do not have physical contact to the environment or the machine itself which prominently known by term machine to machine communication [1]. There will be several sectors that are affected by IoT which include healthcare [2], energy management [3], manufacturing, surveillance [4], retail, smart city, and transportation [5].

In order to maintain the best performance during the data delivery process, some messaging protocols have been implemented widely in the IoT environment. These protocols ensure the data to be sent efficiently by evaluating the power consumption, the delivery speed, the used bandwidth, and reliability. Constrained application protocol (CoAP) was introduced for resolving transmission control protocol (TCP) overhead that occurred during the packet delivery as well as reducing the bandwidth usage by using user datagram protocol (UDP). It imitates the hypertext transfer protocol (HTTP) mechanism by generating a response/request in synchronous [6]. CoAP allows devices to push POST command for sending data to the CoAP server and GET command to retrieve data from the CoAP server. Another messaging protocol that considered as a lightweight protocol is message queuing telemetry transport (MQTT). It implements the publisher/subscriber paradigm in asynchronous. The data is published with a specific topic to the MQTT broker where the MQTT broker can directly forward the data to the subscribers that have already subscribed [6]. Some research has already investigated the effectiveness of these protocols [7]-[16]. Mayub *et al.* [7] proposed an automated system for maintaining a smart home by utilising MQTT and HTTP, which could perform efficiently. Similarly, in [8] the topic was directed to investigate the power consumption of IoT using MQTT, which could last almost ten days indicating that the messaging protocol could minimise the power usage. The work in [9] mentioned the implementation of CoAP for supporting video streaming application since it provides a stateless transport protocol. Rahman *et al.* [9] provide a congestion control mechanism to improve streaming. Larmo *et al.* [10] tried to compare CoAP and MQTT in NB-IoT environment and obtained that both protocols could perform efficiently although MQTT would perform better in a less crowded link. Amaran *et al.* [11] also tried to compare the performance between CoAP and MQTT sensor node for robotic application which concluded that the MQTT could perform 30% better transmission. The other work [12] performed an overhead comparison between WebSocket, CoAP, and MQTT. Sarafov and Seeger [12] find that the MQTT was useful for distributing massive data for the subscriber, CoAP was efficient for sending data with request/respond scheme which did not require a keep-alive connection and WebSocket provided duplex connection which unprovided by CoAP. The work [13] performed quality of service (QoS) over MQTT and CoAP during both low and high traffics. MQTT maintained throughput steadily while suffered packet loss along with the growth of sending rates. CoAP presented the best performance for latency because of implementing UDP. In term of energy efficiency, the work [14] stated that MQTT-SN provided better energy consumption since it has less complexity than CoAP environment. However, in regards to the infrastructure of implementation, CoAP has more uncomplicated requirement because it does not require an additional node for collecting and distributing data. Surprisingly, in [15], [16], the works mentioned that CoAP performed better than the other messaging protocols determined by the latency, energy consumption, throughput, and packet loss. This result could happen because the connectionless transmission between client and server in CoAP architecture delivered less complicated mechanism.

Both protocols are essential in IoT; therefore, it will be useful if there is a system/layer that can provide a bridging mechanism in order to maintain direct communication between the client on different protocol's server. The works in [17]-[25] proposed bridging method for enabling various protocol to connect coherently. Schmitt *et al.* [17] created a bridging mechanism between different MQTT broker/platform using multi-agent system that could create bridging automatically. The results showed 20% more efficient than the static model. Wardana *et al,* [18] developed a platform system that was intended to manage multiple MQTT broker server in the cloud. The system can successfully handle and manage multiple clients from different hardware vendor as well as broker server obtaining a result of response time about 10 seconds. Ibrahim *et al.* [19] proposed a representational state transfer application programming interface (REST API) service for connecting the client with the server as a middleware. The client could send GET and POST message. The middleware was only consuming 74 MB of random-access memory (RAM) rather than the other. Collina *et al.* [20] created a bridging system for REST and MQTT protocol. It provided a bidirectional communication between clients and could handle the data transfer efficiently. The application was developed in a web interface. Mamlook *et al.* [21] built a system based on Windows Communication Foundation (WCF) to translate HTTP and MQTT in duplex mode. The system was developed as a web service using AJAX (Asynchronous JavaScript) to deliver real-time data event from MQTT publishers. Similarly, [22] proposed a system based on web service that acted as a proxy server to translate CoAP – HTTP in a duplex manner. The proxy server could reduce the data size of the message being sent to the web service provider (WSP). Palavras *et al.* [23] introduced secure multi-protocol integration bridge for the IoT (SeMIBIoT) as a translation server using a testbed setup (BeagleBoard XM). The conversion covered HTTP–COAP, HTTP-MQTT, HTTP–extensible messaging and presence protocol (XMPP), MQTT-CoAP, XMPP-CoAP, XMPP-MQTT. Instead of performing specific bridging mechanism, [24] built a system that could translate several protocols including CoAP, MQTT, DDS, and websocket message into HTTP. The end clients send the data to the gateway called MiddleBridge as an intermediary device that connects several distinct clients and transforms the message into HTTP then presents it on the IoT platform. Khaled and Helal [25] proposed a framework that could translate multiple IoT messaging protocols and measured energy consumption. The results showed a low amount of energy usage for the homogenous and heterogeneous scenario. Although some research concerned about bridging in IoT environment have been previously performed, there were no papers that directly mentioned or developed a server which can enable seamless communication between CoAP-MQTT/MQTT-CoAP. Therefore, this paper was directed to create a conversion server for building an intermediary node/server that can translate the message originated from the MQTT/CoAP client in duplex mode. The development of the proposed server may allow existing clients that have different types of messaging protocols (CoAP and MQTT) for communicating without layering restriction. Therefore, the CoAP client can receive event-based data from MQTT publisher while the MQTT subscriber may obtain state transfer/event-based data from CoAP client. The author also proposed analysis method for evaluating the performance of the conversion server by using QoS indicator (packet loss, processing delay) compared to the corresponding references mentioned before.

The remaining content of this paper is presented as follows: section 2 concerns about the proposed research method and description to enable communication between MQTT and CoAP client. In section 3, the authors discussed the research result and analysis in details followed by the research conclusion in section 4.

2. RESEARCH METHOD

The experiment was conducted in OpenFlow environment using the designed topology explained in Figure 1 while the real implementation illustration described in Figure 2. The devices on the topology consist of the software defined network (SDN) controller using RYU [26], the SDN switch using TP-Link router version WDR4300 as the real hardware switch installed by Open vSwitch (OVS) [27] via OpenWrt [28], three mini servers using Raspberry Pi 3 (the conversion server, the MQTT broker, and the CoAP server), and four IoT clients using ESP8266 node microcontroller unit (NodeMCU) module functioned as the MQTT publisher, MQTT subscriber, CoAP GET client, and CoAP POST client respectively. All of the mentioned clients were connected wirelessly while the other node connected using the ethernet cable. OpenFlow became the southbound interface used during the experiment [29]. The conversion server's code and application were written in python environment; therefore, its cloud be deployed in any devices as long as the dependence library was installed.

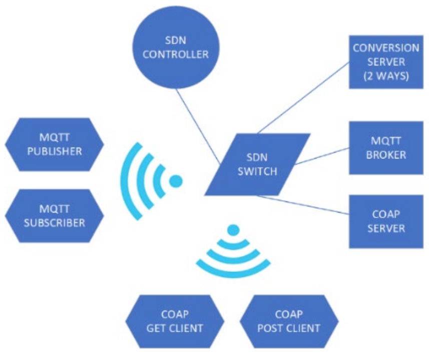

Figure 1. Experiment topology

Figure 2. Hardware used during experiment

Based on the transmission control protocol/internet protocol (TCP/IP) layer abstraction, the proposed method modified the application layer by generating a translation module on the conversion server illustrated in Figure 3. The module can perform two functionalities for translating MQTT data to CoAP and vice versa. In term of MQTT-CoAP translation, the conversion server will generate MQTT subscriber as well as CoAP POST client object to transform the message configuration sent by the MQTT publisher into CoAP data in UDP format.

The data extraction process will allow the conversion server to separate the topic and the published data then convert the parsed data into a CoAP message using the CoAP POST client module. The network administrator could specify the uniform resource locator (URL) for sending the CoAP data. Similarly, the data transformation process from CoAP into MQTT was implemented by CoAP GET client and MQTT publisher object. The server retrieves the recent data based on specific URL using the CoAP GET client; then the server will extract the message and forward it into MQTT publisher object. Subsequently, the data will be delivered into MQTT broker through a publisher object.

In default, the RYU controller implements the learning switch algorithm for resolving the packet forwarding on the data plane layer. The controller would send the OpenFlow flow table modifications packet (OFPT_FLOW_MOD) message to the SDN switch for installing the flow rule for filtering as well as performing the forwarding function based on the media access control (MAC) address learned by the controller [30]. The conversion process was modulated into two distinct subprocess which handled the translation of MQTT-CoAP and CoAP-MQTT. Figure 4 explains the procedural step to convert the MQTT message into a CoAP message and send it to the CoAP server.

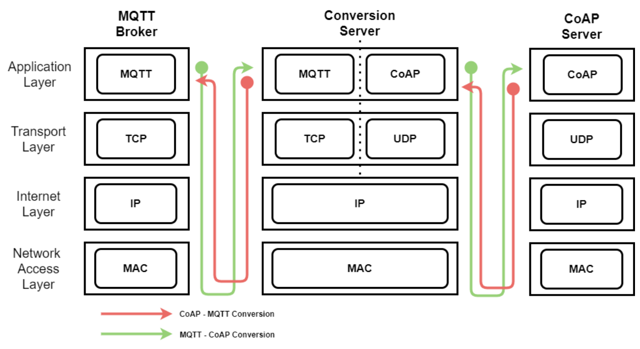

Figure 3. TCP/IP layer abstraction

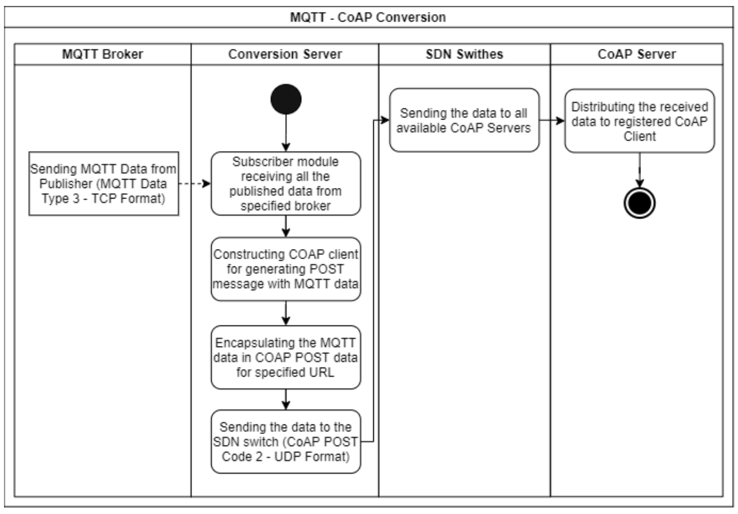
MQTT - CoAP Conversion
MQTT Broker
Conversion Server
SDN Swithes
CoAP Server
Sending MQTT Data from Publisher (MQTT Data Type 3 - TCP Format)
Subscriber module receiving all the published data from specified broker
Constructing COAP client for generating POST message with MQTT data
Encapsulating the MQTT data in COAP POST data for specified URL
Sending the data to the SDN switch (CoAP POST Code 2 - UDP Format)
Sending the data to all available CoAP Servers
Distributing the received data to registered CoAP Client

Figure 4. Activity diagram of MQTT-CoAP conversion

The initial step is giving the MQTT-CoAP module the detail address and port of the available MQTT broker and CoAP server that want to be translated which include the MQTT broker address, MQTT broker port, MQTT topic to be subscribed, CoAP server address, CoAP server port, and the URL for sending the data to CoAP server. All of the input variables can be initiated through a command-line interface. Subsequently, the server generates an MQTT subscriber object in order to receive the actual data from the publisher based on the specified topic, which can be extended into multiple topics. Upon receiving the data from the publisher, the server parses the data structure and extracts the packet's topic and also the packet's payload/data. The next step is building CoAP client object for sending MQTT data encapsulated in CoAP POST message. The data is sent directly to the CoAP server after defined the specific URL that has been mentioned previously. The regular CoAP client can directly retrieve the most recent message from the MQTT publisher with the same URL provided. In order to differentiate the data, there was an additional header on the payload for indicating the data originated from MQTT publisher.

Similarly, the CoAP-MQTT translation process depicted from Figure 5 was initiated by developing CoAP client object that could perform GET command for the specified URL, retrieving the most recent data from the CoAP client (ESP8266 NodeMCU). The conversion module for CoAP-MQTT has several input variables which include the CoAP server's address, CoAP server's port, CoAP server's URL, MQTT broker's address, and MQTT broker's port. After receiving the data from specific URL in CoAP server, the CoAP object in this module would parse the packet for getting data; then the server would generate MQTT client object as publisher for sending the CoAP data directly to the MQTT broker. The message component also includes a specific topic for determining the data generated by the CoAP Client. The experiment was analysed by calculating the processing delay, round trip time (RTT) value, the conversion loss, Core power consumption, and the central processing unit (CPU) usage of the SDN controller and the conversion server. The experiment consists of two scenarios which include sending 200 packets data originated from the client in the rate of 10 packet/seconds and 20 packets/seconds. The packet size for both scenarios was 200 KB.

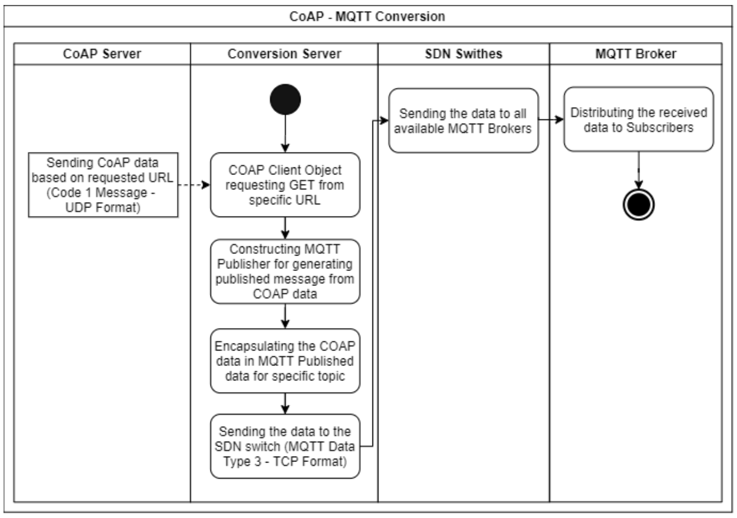
CoAP - MQTT Conversion
CoAP Server
Conversion Server
SDN Swithes
MQTT Broker
Sending CoAP data based on requested URL (Code 1 Message - UDP Format)
COAP Client Object requesting GET from specific URL
Constructing MQTT Publisher for generating published message from COAP data
Encapsulating the COAP data in MQTT Published data for specific topic
Sending the data to the SDN switch (MQTT Data Type 3 - TCP Format)
Sending the data to all available MQTT Brokers
Distributing the received data to Subscribers

Figure 5. Activity diagram of CoAP-MQTT conversion

3. RESULTS AND ANALYSIS

The research results are extracted from two distinct scenarios by generating packet data massively from either MQTT publisher or CoAP client. The packets were delivered at rate ten packets/second and 20 packets/second respectively. In order to measure the effectiveness of the conversion server, several variables were captured including the processing delay of the conversion, packet loss during the conversion, RTT value, Core's power consumption, and the CPU usage of the controller and the conversion server. Table 1 shows the average delay of the conversion process between the messaging protocol. The conversion process of MQTT message translated into CoAP message required 52 and 53 second for each of the experiment scenarios.

Table 1. The average conversion delay

	MQTT-CoAP		CoAP-MQTT	
	10ps	20ps	10ps	20ps
Average Conversion Delay (s)	52.43407	53.11167	0.179852	0.182637

The value occurred significantly because the conversion server was overwhelmed by the published data sent by the broker. The growth of the delay is illustrated in Figure 6 and Figure 7. The alteration of the delay dynamically increased along with the number of data sent to the broker. This trend could occur because the broker did not efficiently send the data from the publisher steadily. During the experiment, the broker would send all of the incoming packets reactively to all subscribers as long as the link were not congested. The maximum number of packets sent by the broker for both scenarios were 20 packets in second. Therefore, the conversion server would obtain the incoming packet more than one packets in second which forced the server to perform queueing of the data from the MQTT broker as well as generating the CoAP client object at the same time affecting the conversion process requiring longer time.

A similar pattern also occurred in the 20 packets per second scenario where the time to perform the conversion process increased dynamically along with the growth of data sent by the publisher. In converse, the CoAP-MQTT translation process did not suffer from the queueing because CoAP was not designed for an event-based protocol which did not require persistence connection (connectionless). Therefore, in order to receive the up-to-date information from the CoAP server, the conversion server has to recreate CoAP GET client object for retrieving new data from the CoAP server. Depicted from Figure 8 and Figure 9, the graphics illustrate the conversion delay for the incoming packets in CoAP-MQTT. The delay is pointed below 0.4 seconds for both scenarios indicating that there was no queue during the conversion process. However, the significant part is described on the xtics of the graph. The recorded data did not mention all of the incoming packets, because some of the packets were lost due to waiting mechanism on conversion server. On the experiment, the authors set the GET command interval process every one second, which generated no queue and increased the packet loss value.

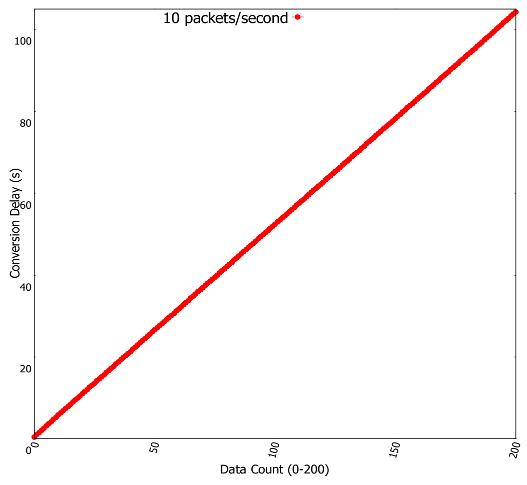

Figure 6. The growth of MQTT-CoAP conversion during 10ps

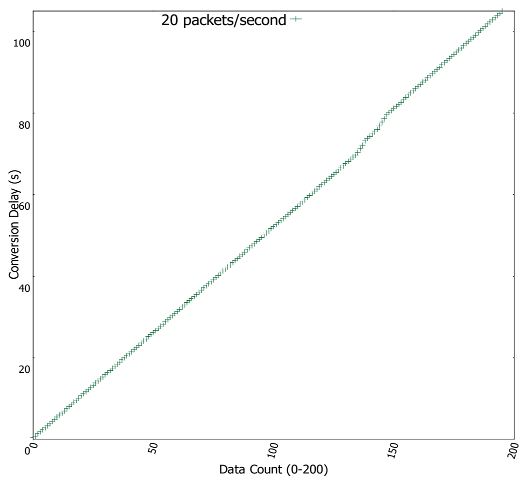

Figure 7. The growth of MQTT-CoAP conversion during 20ps

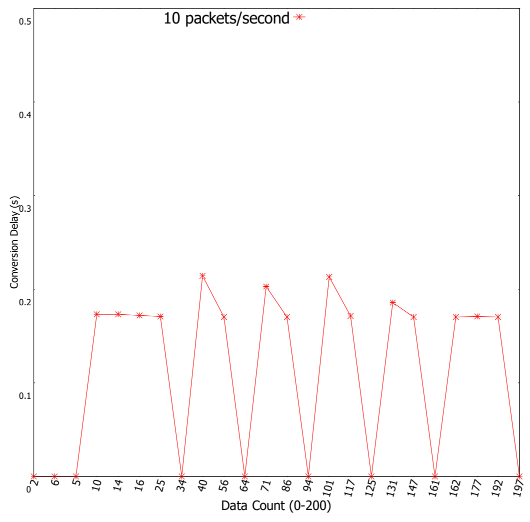

Figure 8. The growth of CoAP-MQTT conversion during 10ps

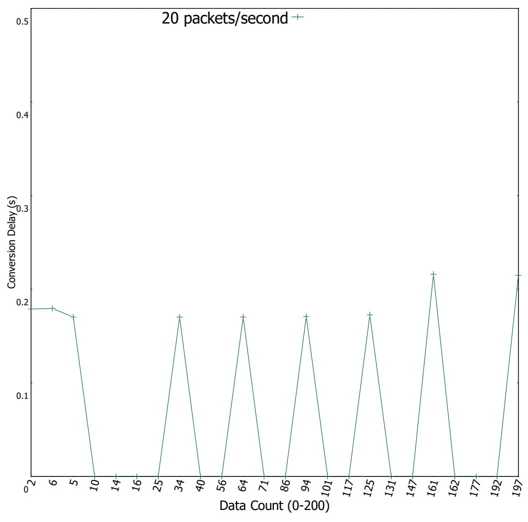

Figure 9. The growth of CoAP-MQTT conversion during 20ps

The conversion delay also affected the value of RTT in average. The average RTT for both scenarios were pointed more than the average value of conversion delay. The RTT value was not significantly changed even though there was an overwhelming request originated from the client depicted from Table 2. The number of conversion loss in percentage is described in Table 3. There was no unprocessed packet during the MQTT-CoAP conversion because all of the incoming packets were received and processed. However, along with the growth of the received data, the value of the conversion delay also increased. The conversion loss value for CoAP -MQTT translation almost reached 92% because there was an interval of waiting time for the GET object in conversion server. This pattern also occurred because the initiation of the publisher object required more steps, including the TCP three-way handshake process, the connect, and publish commands. In contrary, the MQTT-CoAP conversion produced less packet overhead because the CoAP was transmitted using UDP. Moreover, the code structure for CoAP-MQTT translation only utilised a single thread process and client-server architecture for executing the GET command, which affected the real-time data extraction process.

Table 2. The Average RTT value

	MQTT-CoAP		CoAP-MQTT	
	10ps	20ps	10ps	20ps
Average RTT (s)	52.9	53.5	0.1	0.1

Table 3. The conversion loss during experiment

	MQTT-CoAP		CoAP-MQTT	
	10ps	20ps	10ps	20ps
Conversion Loss (%)	0	0	92.53731	95.52239

In term of the Core Power consumption of the conversion server illustrated in Table 4, the average voltage's usage was pointed between 1.24-1.25 volts for all scenarios indicating that the translation process did not consume a large amount of energy. The CPU usage for both controller and the conversion server depicted from Table 5 and Table 6. the message translation procedure did not consume a vast amount of computing resources even though the code for the application were written in python code. In average, the total CPU consumption during the experiment for both controller and the conversion server required below than 1% indicating that the process did not demand special computational resources which acquired better performance rather than [23] which utilised more than 5% CPU usage for the translation scenarios.

Compared to the previous related research [23], [24], even though this paper utilised OpenFlow environment as packet filtering process the value of single packet conversion process was nearly equal around 0.5 s for the MQTT-CoAP and 0.1 s for CoAP-MQTT translation process which was better than the result from paper [23], [24]. In term of the packet size, the packet transmission process using 200 KB also did not affect the RTT value compared to the other paper, [23] using 16 bytes, [24] using 23 bytes.

Table 4. Conversion server's core power consumption

	MQTT-CoAP		CoAP-MQTT	
	10ps	20ps	10ps	20ps
Core Voltage (Volts)	1.24	1.25	1.24	1.24

Table 5. The CPU usage of conversion server

	MQTT-CoAP		CoAP-MQTT	
	10ps	20ps	10ps	20ps
CPU Usage (%)	0.3	0.3	0.13	0.14

Table 6. The CPU usage of controller

	CoAP-MQTT		MQTT-CoAP	
	10ps	20ps	10ps	20ps
CPU Usage (%)	0.3	0.3	0.3	0.3

4. CONCLUSION

Developing a bridging/conversion server is essential for providing bidirectional communication between CoAP and MQTT client. In conclusion, the conversion mechanism can be implemented successfully. Both the CoAP server and MQTT broker can receive the translation message from both directions. However, there should be a mechanism that control which topic or data that want to be translated in order to prevent the massive number of packets entering the conversion server and packet loss event. Moreover, the utilisation of subprocess during the conversion codes consumed more time during the translation of CoAP-MQTT, resulting in a high rate of conversion loss. These problems can be fixed as long as the sensor/things nodes performing the periodical update and implementing multithreading. For further reference, the authors will investigate the possibility of developing an application-based solution embedded in an SDN controller for achieving network automation in the IoT environment which can be elaborated to managing the networking environment on several layers (application, transport, internet, and data link). The deployment of the application-based solution installed in the SDN controller may reduce the processing delay as well as minimise the topology complexity by omitting the conversion server node. The application may have a capability to thoroughly analyse the incoming IoT packet, convert the IoT packet, and distribute the converted packet through packet-out message (OFPT_PACKET_OUT) service available in OpenFlow standard.

ACKNOWLEDGEMENTS

The authors would like to express profound gratitude to Universitas Muhammadiyah Malang and Laboratorium Informatika Universitas Muhammadiyah Malang for giving direct support and facilities during the experiment.

REFERENCES

[1] M. Gunturi, H. D. Kotha, and M. Srinivasa Reddy, "An overview of internet of things," *Journal of Advanced Research in Dynamical and Control Systems*, vol. 10, no. 9, pp. 659-665, 2018.

[2] W. Taylor, S. A. Shah, K. Dashtipour, A. Zahid, Q. H. Abbasi, and M. A. Imran, "An Intelligent Non-Invasive Real-Time Human Activity Recognition System for Next-Generation Healthcare," *Sensors*, vol. 20, no. 9, pp. 2653-(1-20), May 2020, doi: 10.3390/s20092653.

[3] A. R. V. B. R. Selvam, T. Anuradha, S. Jayakumar, P. M. Venkatesh, "IoT based Energy Management for Hybrid Solar and Wind Energy System," *International Journal of Future Generation Communication and Networking*, vol. 13, no. 2, pp. 426-438, 2020. [Online]. Available: http://www.sersc.org/journals/index.php/IJFGCN /article/view/12017/6298.

[4] D. Ciuonzo and P. Salvo Rossi, "DECHADE: DEtecting slight Changes with HArd DEcisions in Wireless Sensor Networks," *Int. J. Gen. Syst.*, vol. 47, no. 5, pp. 535-548, Jul. 2018, doi: 10.1080/03081079.2018.1455192.

[5] F. G. Abdulkadhim, Z. Yi, C. Tang, M. Khalid, and S. A. Waheeb, "A survey on the applications of IoT: An investigation into existing environments, present challenges and future opportunities," *TELKOMNIKA Telecommunication Computing Electronics and Control*, vol. 18, no. 3, pp. 1447-1458, 2020, doi: 10.12928/TELKOMNIKA.v18i3.15604.

[6] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate, "A Survey on Application Layer Protocols for the Internet of Things," *Trans. IoT Cloud Comput.*, vol. 3, no. 1, pp. 11-17, 2015, doi: 10.5281/ZENODO.51613.

[7] A. Mayub, Fahmizal, M. Shidiq, U. Y. Oktiawati, and N. R. Rosyid, "Implementation smart home using internet of things," *TELKOMNIKA Telecommunication Computing Electronics and Control*, vol. 17, no. 6, pp. 3126-3136, 2019, doi: 10.12928/TELKOMNIKA.v17i6.11722.

[8] V. Kanakaris, G. A. Papakostas, and D. V. Bandekas, "Power consumption analysis on an IoT network based on wemos: A case study," *TELKOMNIKA Telecommunication Computing Electronics and Control*, vol. 17, no. 5, pp. 2505-2511, 2019, doi: 10.12928/TELKOMNIKA.v17i5.11317.

[9] W. U. Rahman, Y. S. Choi, and K. Chung, "Performance Evaluation of Video Streaming Application over CoAP in IoT," *IEEE Access*, vol. 7, pp. 39852-39861, 2019, doi: 10.1109/ACCESS.2019.2907157.

[10] A. Larmo, A. Ratilainen, and J. Saarinen, "Impact of coAP and MQTT on NB-IoT system performance," *Sensors*, vol. 19, no. 1, 2019, doi: 10.3390/s19010007.

[11] M. H. Amaran, N. A. M. Noh, M. S. Rohmad, and H. Hashim, "A Comparison of Lightweight Communication Protocols in Robotic Applications," *Procedia Comput. Sci.*, vol. 76, no. Iris, pp. 400-405, 2015, doi: 10.1016/j.procs.2015.12.318.

[12] V. Sarafov and J. Seeger, "Comparison of IoT Data Protocol Overhead," *Network Architectures and Services*, pp. 7-14, 2018, doi: 10.2313/NET-2018-03-1.

[13] F. Oukasse and S. Rakrak, "A Comparative Study of MQTT and CoAP Application Layer Protocols via. Performances Evaluation," *Journal of Engineering and Applied Sciences*, vol. 13, no. 15. pp. 6053-6061, 2018, doi: 10.36478/jeasci.2018.6053.6061.

[14] M. Martí, C. Garcia-Rubio, and C. Campo, "Performance Evaluation of CoAP and MQTT_SN in an IoT Environment," *Proceedings*, vol. 31, no. 1, p. 49, 2019, doi: 10.3390/proceedings2019031049.

[15] A. Larmo, F. Del Carpio, P. Arvidson, and R. Chirikov, "Comparison of CoAP and MQTT performance over capillary radios," *2018 Glob. Internet Things Summit, GIoTS 2018*, 2018, pp. 1-6, doi: 10.1109/GIOTS.2018.8534576.

[16] T. Moraes, B. Nogueira, V. Lira and E. Tavares, "Performance Comparison of IoT Communication Protocols," *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 2019, pp. 3249-3254, doi: 10.1109/SMC.2019.8914552.

[17] A. Schmitt, F. Carlier, and V. Renault, "Dynamic bridge generation for IoT data exchange via the MQTT protocol," *Procedia Comput. Sci.*, vol. 130, pp. 90-97, 2018, doi: 10.1016/j.procs.2018.04.016.

[18] A. A. Wardana, A. Rakhmatsyah, A. E. Minarno, and D. R. Anbiya, "Internet of Things Platform for Manage Multiple Message Queuing Telemetry Transport Broker Server," *Kinet. Game Technol. Inf. Syst. Comput. Network, Comput. Electron. Control*, vol. 4, no. 3, pp. 197-206, 2019, doi: 10.22219/kinetik.v4i3.841.

[19] A. K. M. Ibrahim, R. A. Rashid, A. H. F. A. Hamid, M. Adib Sarijari, and M. A. Baharudin, "Lightweight IoT middleware for rapid application development," *TELKOMNIKA Telecommunication Computing Electronics and Control*, vol. 17, no. 3, pp. 1385-1392, 2019, doi: 10.12928/TELKOMNIKA.V17I3.11793.

[20] M. Collina, G. E. Corazza and A. Vanelli-Coralli, "Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST," *2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC)*, 2012, pp. 36-41, doi: 10.1109/PIMRC.2012.6362813.

[21] R. Mamlook, O. F. Khan, and T. S. Mohammed, "Bridging for cross protocol talk in IOT devices using windows communication foundation," *Int. J. Eng. Technol.*, vol. 7, no. 3.29, pp. 196-198, 2018, doi: 10.14419/ijet.v7i3.29.18794.

[22] W. Jin and D. Kim, "Development of virtual resource based IoT proxy for bridging heterogeneous web services in IoT networks," *Sensors*, vol. 18, no. 6, 2018, doi: 10.3390/s18061721.

[23] E. Palavras, K. Fysarakis, I. Papaefstathiou and I. Askoxylakis, "SeMIBIoT: Secure Multi-Protocol Integration Bridge for the IoT," *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1-7, doi: 10.1109/ICC.2018.8422486.

[24] M. A. A. da Cruz, J. J. P. C. Rodrigues, P. Lorenz, P. Solic, J. Al-Muhtadi, and V. H. C. Albuquerque, "A proposal for bridging application layer protocols to HTTP on IoT solutions," *Futur. Gener. Comput. Syst.*, vol. 97, no. 2019, pp. 145-152, 2019, doi: 10.1016/j.future.2019.02.009.

[25] A. E. Khaled and S. Helal, "Interoperable communication framework for bridging RESTful and topic-based communication in IoT," *Futur. Gener. Comput. Syst.*, vol. 92, pp. 628-643, 2019, doi: 10.1016/j.future.2017.12.042.

[26] RYU Project Team. (Dec. 2016). RYU SDN Framework. [Online]. Available: https://book.ryu-sdn.org/en/Ryubook.pdf
[27] B. Pfaff *et al*., "The design and implementation of open vSwitch," in *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2015*, 2015, pp. 117-130.
[28] C. E. Palazzi, M. Brunati, and M. Roccetti, "An OpenWRT solution for future wireless homes," in 2010 IEEE International Conference on Multimedia and Expo, 2010, pp. 1701-1706, doi: 10.1109/ICME.2010.5583223.
[29] B. Heller *et al*., "OpenFlow Switch Specification", Version 1.3.0 (Wire Protocol 0x04)," pp. 1-105, 2012. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf
[30] A. C. Jaramillo, R. Alcivar, J. Pesantez and R. Ponguillo, "Cost Effective test-bed for Comparison of SDN Network and Traditional Network," *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*, 2018, pp. 1-2, doi: 10.1109/PCCC.2018.8711223.

BIOGRAPHIES OF AUTHORS

Fauzi Dwi Setiawan Sumadi achieved his bachelor degree in 2014 at Bandung Telkom University. He studied informatics engineering and focused for exploring the implementation of wireless sensor network. Subsequently, in 2017 he graduated from the University of Queensland Australia for his master degree program in computer science which focused to analyse the vulnerability in software defined network. Nowadays, he becomes one of the main lecturers in Informatics Department at Universitas Muhammadiyah Malang and maintains his research in the implementation of artificial intelligence in computer network, distributed computing, and the SDN.

Agus Eko Minarno graduated with Master of Computer from Sepuluh Nopember Technological Institute (ITS), Surabaya. Currently he is a senior lecturer in Informatics Department University of Muhammadiyah Malang (UMM) and active as the head of Informatics laboratory in UMM. His area of interest is machine learning, deep learning, image processing, data mining, knowledge representation, pattern recognition.

Lailis Syafaah has passed her Bachelor and Master degree in Electrical Engineering from Sepuluh Nopember Technological Institute (ITS). She has successfully done her Doctoral program for Medical Electronic from University of Brawijaya. Currently she as senior lecturer in EE Dept Universitas Muhammadiyah Malang and actively doing research for Electrical Engineering, Sustainable Energy and Medical Electronic.

Muhammad Irfan was born in Mojokerto, Indonesia in 1966. He graduated with Bachelor of Informatics Engineering from Sepuluh Nopember Technological Institute (ITS), Surabaya. His Master was also from the same institute gradated on 2000. Currently he is a senior lecturer in Electrical Engineering Department Universitas Muhammadiyah Malang (UMM), continuing his doctoral degree at the University of Technology Malaysia, and active members of Research and Development Institute of UMM. His area of interest is Electrical Power Engineering, Power Electronics & Drive, Digital Signal Processing and Energy Efficiency.