JITECS ID 101

by 101 Jitecs

Developing Actor-Based Middleware as Collector System for Sensor Data in Internet of Things (IoT)

Abstract

The use of Internet of Things (IoT) plays an important role in supporting wireless communication for middleware in collecting data sensors. An actor-based middleware is designed to bridge protocol differences between cloud and sensor nodes. This middleware also acts as an initiator in accessing data from several sensor nodes, and then sending data that has been collected to the cloud. Incorporating the differences of communication protocols and data formats between sensor nodes and cloud is the responsibility of middleware. This Middleware acts as an actor by acting proactively accessing data from each sensor node, so that it can facilitate the completion of sending data from the sensor node to the middleware by avoiding from "signal collisions" among sensor nodes. After the data is collected in the middleware, the data is sent to the cloud using the Websocket or HTTP protocol above the TCP / IP protocol. The performance of the system is evaluated based on the success of the middleware bridging communication between sensor nodes and the cloud, as well as the readability of IoT data sensors that have been adjusted by cloud. The test results show that built-in middleware can bridge protocols between cloud and sensor nodes. In addition, the Websocket usage protocol produces a lower delay value than the MQTT and CoAP protocols.

1. Introduction

The Internet of Things (IoT) paradigm allows connectivity between various types of devices such as household furniture, environmental sensors, cameras, motorized vehicles, etc., using Internet media. IoT is simply a concept in which things in life can be equipped with sensors, microcontrollers, actuators, and transceiver modules and protocol stack, so that they can communicate with each other and interact with the surrounding environment (Atzori et al., 2010).

An IoT-based system can consist of several components, including components of identification, sensing, computing, communication, service, and semantics. The main function of the computational component is processing data generated by the sensing component. Light computing processes such as converting analog signals from sensors to digital signals can be done on microprocessors / minicomputers located on the sensor node. Data processing can also be done in data centers (data centers) with the concept of cloud computing (cloud computing), which can support processing of more complex and larger data. Data processing carried out in the cloud can overcome the problem of limited electricity resources, memory and computing capabilities and service capabilities on the sensor node.

Sensing data from the sensor node will be sent to the cloud for processing. Because the format of data from sensor nodes cannot always be directly understood by data processors in the cloud due to the diversity of communication protocols used by sensor nodes, middleware is needed to bridge communication between sensor nodes and the cloud (Rahmani et al., 2015). Because IoT is composed of intelligent devices in large numbers, one of the challenges in trying to maintain the availability of connectivity between these devices is interoperability between the elements involved (Farsi, 2015). The change in the paradigm on the Internet from "interconnected computers" to "interconnected objects" requires no small amount of effort, including the need for a middleware system that can simplify the development of applications and services needed (Chaqfeh & Mohamed, 2012).

A middleware can provide an abstract layer between infrastructure and applications (Atzori et al., 2010). Middleware is capable of integrating applications and services, each of which runs on a heterogeneous system (Mahmoud, 2005). There are several types of IoT middleware architecture, one of which is an actor-based framework, which emphasizes plug and play IoT architecture that is open (Ngu et al., 2017).

In this study, researchers proposed the title "Designing an Actor-based Middleware as an Internet of Things Sensor Data Collection System". The proposed data collection system is expected to bridge the differences in systems that exist in the cloud and sensor nodes. This Middleware will also act as an initiator to access data directly to several sensor nodes, then the middleware sends data that has been collected to the cloud. The difference between communication protocols and the different data formats between sensor nodes and the cloud is the responsibility of the middleware to synchronize. The advantages of the middleware can reduce data traffic from the sensor node directly to the cloud.

Previous research has been able to bridge sensor node protocols with protocols recognized by the cloud, except that data traffic from the sensor node is sent to the middleware and immediately forwarded to the cloud in real-time (Anwari, 2017), so there is a potential for collisions. data traffic that is passing through the communication media. This actor-based middleware will act proactively in accessing data from each sensor node reached by adjusting the protocol supported by the sensor node. This proactive method can prevent each sensor node from sending data simultaneously which results in delivery failure due to a "collision" of data transmission. After the data is collected, data is sent to the cloud with other protocols used, generally with TCP / IP. Sending data from the sensor node to the cloud will be carried out periodically by the middleware. Especially for data with a certain value, the middleware can forward it to the cloud as soon as it is received from the sensor node, without waiting for the scheduled period of sending data from middleware to the cloud. The advantage of this proactive method is to prevent or reduce the problem of data transmission failure. Illustrations of the whole system are shown in Figure 1.1.

The performance of the system created will be evaluated based on the success of bridging communication between sensor nodes and the cloud, as well as the success of readability of IoT sensor data that has been adjusted by cloud.

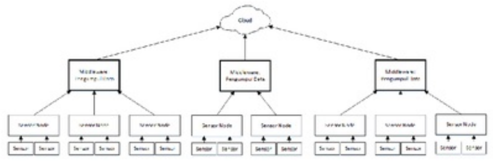


Figure 1.1. Sistem IoT : *Sensor node* IoT – *Middleware - Cloud*

Based on the background, the problems in this study can be formulated as follows:

1. How can the IoT sensor data collection system (middleware) access data from the sensor node?

2. How does the IoT sensor data collection system (middleware) authenticate and authorize the sensor node?

3. How can the IoT sensor data collection system (middleware) send data to the cloud that has a different communication protocol?

4. How can the IoT sensor data collection system (middleware) send data to the cloud based on time periods and based on data values?

5. How is the performance of the IoT sensor data collection system (middleware) seen from the functionality and quality of service?

2. Literature Review

In the study of this literature discussed earlier research related to IoT middleware as an intermediary between several different protocols. In "Challenges in Middleware Solutions for the Internet of Things" (Mohamed, 2012) there are some challenges faced by IoT middleware. One of the challenges is interoperability. Interoperability is one of the biggest challenges of IoT middleware, because there are so many and varied devices that interact with IoT, both now and in the future.

In "IoT Middleware Development as a Gateway for CoAP, MQTT, and Websocket Protocols", sensor node protocols with protocols recognized by the cloud can be bridged, except that data traffic from the sensor node is sent to the middleware and directly forwarded to the cloud in real-time. time (Anwari, 2017; OASIS, 2014; HiveMQ, 2015; IETF, 2016). This raises the potential for collisions / collisions between data traffic that is passing through the communication media.

The CoAP protocol and WebSocket operate in the form of a "client-server". The server will serve connection requests from other parties, namely the client. Client as the initiator who will contact the server. The server must have a specific IP address where the client can contact him. As described in the previous sub-chapter.

Unlike the two protocols above, MQTT operates in a "publish / subscribe" manner as explained in the MQTT sub-chapter. There are parties as subscribers, who subscribe (a topic) to brokers. One party as a publisher (a topic), which will send / publish through a third party, the broker. As a third party, brokers can disseminate information to each subscriber. Unlike the client-server, subscribers and publishers do not need to know their respective IP addresses. However, subscribers and publishers must contact brokers so that subscribers and publishers can exchange information. For this reason, the broker's address must be known by the subscriber and publisher. The number of subscribers and publishers is not limited, as long as they can be handled by brokers.

Following the next development, the broker at MQTT needed a database to store the data received from the publisher and then sent to all subscribers. Databases that can support the functionalities of MQTT are usually Redis and MongoDB. The data can be sent or accessed / distributed based on certain topics. Only with the appropriate topic, the subscriber will accept it.

The following example is how communication runs between publishers and subscribers. Nodes that as Subscriber register themselves with Topic1, then when a node publishes with Topic1 the subscriber nodes will be informed. So the broker receives information based on Topic1, and will disseminate that information to the nodes that subscribe to Topic1.

2.1. IoT Middleware

In the context of IoT, the existence of middleware can bridge communication between sensor nodes and clouds, because the format of data from sensor nodes cannot always be directly understood by data processors in the cloud due to the diversity of communication protocols used by sensor nodes (Rahmani et al. 2015).

Middleware for IoT is needed for several reasons, including:

1. The difficulty of forming and forcing vendors to use a standard on a variety of devices connected in the IoT network.
2. The need for an abstract layer can be used on all types of devices.
3. Middleware provides an Application Programming Interface (API) for communication with the physical layer and provides services to applications.
4. Reducing data traffic directly from the sensor node to the cloud.

A middleware can provide an abstract layer between infrastructure and applications (Atzori et al., 2010). There are several types of IoT middleware architecture, one of which is an actor-based framework, which emphasizes plug and play IoT architecture that is open (Ngu et al., 2017). While according to Qusay Mahmouh (2005) middleware can integrate several systems that run heterogeneous applications and services.

3. System Design

System design in this study consists of system flow design, middleware design, application design, and network architecture design for testing. System design is made by describing the system architecture based on the results of analysis of hardware requirements and software requirements that have been determined. This design is also based on the theory obtained in the literature study in the form of middleware architecture that supports interoperability. The design of this system is made to facilitate implementation, testing and analysis.

The built-in system as in Figure 3.1 can represent the whole system.

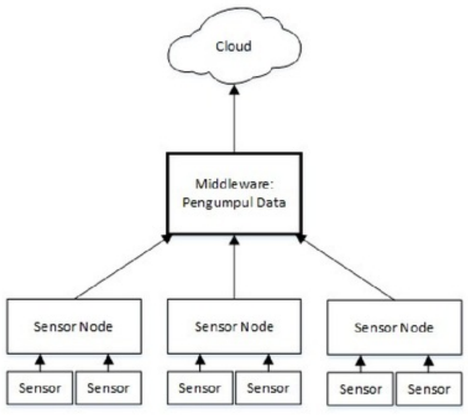


Figure 3.1. *Middleware* System to Bridge Protocol Differences

Meanwhile, the functional needs for this actor-based middleware are as follows

1. Sensor node-sensor nodes that use different communication protocols, namely CoAP, MQTT and WebSocket. Each sensor node is considered able to access the sensor devices installed on it, such as temperature sensors, humidity, and so on. Also each sensor node is equipped with a communication protocol and wireless device to send the data to the middleware.

2. Cloud uses only one type of communication protocol, in this study the cloud will use WebSocket.

3. Middleware can choose the protocol that corresponds to that used by the sensor node-sensor node to communicate. Middleware will read data from each sensor node sequentially (round robin). Doing round robin avoids from collision of wireless communication among sensor nodes.

4. Middleware can arrange data from each sensor node that has been read, and prepare by combining all data from each sensor node to be sent to the cloud.

5. Middleware uses the WebSocket protocol to send data that has been collected from the sensor nodes underneath.

6. As a support, the cloud can process and display combined data sent by middleware.

3.1 System Flow Designing

In the system developed, there are two types of data transmission. The first shipment is data from the sensor node (as a data source) to the middleware. The second shipment is data from middleware to the cloud (as a data store from all sensor nodes). Middleware can proactively access data from sensor nodes. The detailed mechanism of communication of each sensor node with different protocols is shown in Figure 3.2 and devices are with allocated IP addresses.

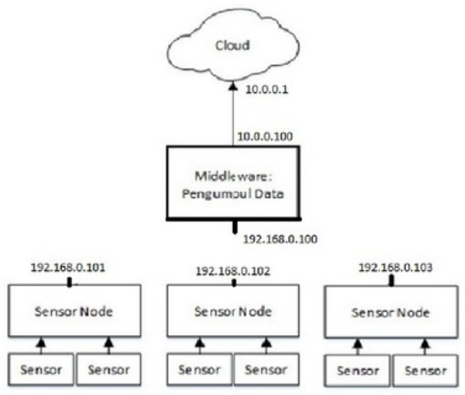


Figure 3.2. Middleware and Other Devices with IP Addresses

3.2 Designing Middleware

Middleware developed applies a publish / subscribe pattern. This Middleware consists of three parts: application gateway, service unit, and sensor gateway. Application gateways provide an interface for middleware to send data to the cloud. The service unit is a part of the middleware that serves to provide an interface for the sensor gateway and application gateway. This service unit connects sensor nodes with brokers, and connects the cloud with brokers. The gateway sensor provides an interface for the middleware to read data from IoT node sensors. The gateway sensor makes it possible to use different communication protocols, and this protocol is adapted to the protocol used by the sensor node device.

Not only bridging sensor node protocols with protocols recognized by the cloud, middleware designed in this study will act proactively in accessing data from each sensor node reached by adjusting protocols supported by sensor nodes. This proactive method can prevent each sensor node from sending data simultaneously which results in delivery failure due to a "collision" of data transmission. After the data is collected, data is sent to the cloud with other protocols used, generally with TCP / IP. Sending data from the sensor node to the cloud will be carried out periodically by the middleware. Especially for data with a certain value, the middleware can forward it to the cloud as soon as it is received from the sensor node, without waiting for the scheduled period of sending data from middleware to the cloud. The advantage of this proactive method is to prevent or reduce the problem of data transmission failure.

Hence, this middleware device is designed to have three parts. The internal parts of the middleware along with the protocol or function in each part are shown in Figure 3.3.

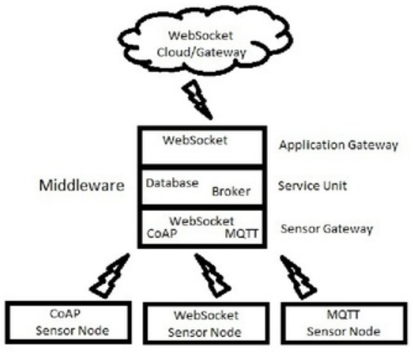


Figure 3.3. Internal Parts of Middleware Systems and their Functions

Sensor gateways in middleware use protocols that are in accordance with the protocol used by the sensor node. Therefore, the communication protocol registration process from each sensor node in the middleware is needed.

The sequences diagram of communication between middleware and sensor nodes with different protocols, and between middleware and cloud are shown in Figure 3.4, Figure 3.5 and Figure 3.6. It also describes of how the middleware operates to accessing data from all sensor nodes with different protocols, and then to delivering all collected data to cloud.

Nevertheless, before system is running every sensor node's address/ ID must be registered to middleware. Middleware can recognize every node and its in-use protocol.

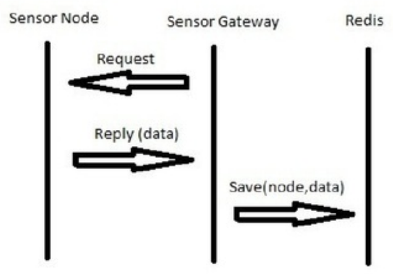


Figure 3.4. Middleware Sequence Diagram to Sensor node with CoAP or WebSocket (Gateway Sensor).

By round robin, middleware directly accesses to every available and registered sensor node provided with either CoAP or WebSocket. Each sensor node can inform middleware its data. Each sensor node with this protocols has to wait for Middleware to access and eventually reply with its data.

The data collected by middleware are stored in data base of Redis (database broker)

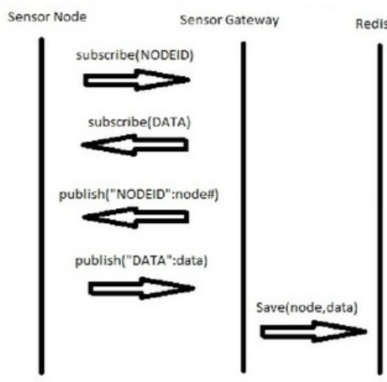


Figure 3.5. Sequence Middleware Diagram to Sensor node with MQTT (Gateway Sensor).

While for a sensor node with MQTT, sensor node needs to send "subcribe" to middleware to inform its existance, and middleware also sends subcribes to sensor node. Subcription messages allow both devices to publish its data. The subcription of middleware to sensor enables to trigger sensor node to send its data, while sensor node is prohibited to proactively send its data until it gets a trigger of "publish message" from middleware. Thus, the natural characterics of sensor node with MQTT by sending or publishing its data by itself is restricted.

The data collected by middleware are stored in data base of Redis (database broker)

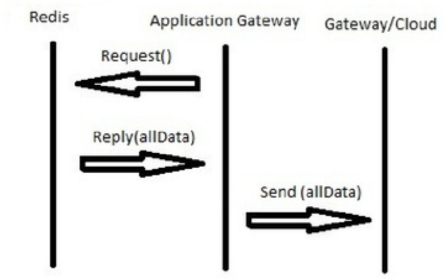


Figure 3.6. Middleware Sequence Sequence to the Cloud with WebSocket (Application Gateway

After getting data from all sensor nodes, eventually the middleware composes and deliveres all data under Redis to the cloud using WebSocket protokol.

3.3. System Requirements

The middleware and sensor nodes are forms of

a. Hardware
   i) Raspbian Jessie
   ii) NodeMCU
b. Software
   i) Framework of Node.js
   ii) Database broker Redis
   iii) Firmware LUA with protocol MQTT, CoAP or WebSocket.

4. Result and Analisis

4.1. Integration Testing

This test is done to get information whether a component can interact with each other when performing certain functions. The test uses two clients, by way of: client A with a protocol subscribing to a particular topic, then client B with another protocol publishing information with the topic, then testing whether client A can receive information on the topic.
The results are shown in Figure 4.1 and Figure 4.2 which represent integration testing with different protocols.

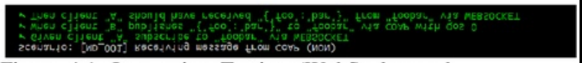

```
✔ Then client "A" should have received "{"foo":"bar"}" from "foobar" via WEBSOCKET
✔ When client "B" publishes "{"foo":"bar"}" to "foobar" via COAP with QoS 0
✔ Given client "A" subscribe to "foobar" via WEBSOCKET
Scenario: [MD_001] Receiving message from CoAP (NON)
```

Figure 4.1. Integration Testing (WebSocket and CoAP)

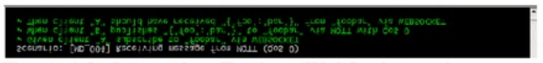

```
✔ Then client "A" should have received "{"foo":"bar"}" from "foobar" via WEBSOCKET
✔ When client "B" publishes "{"foo":"bar"}" to "foobar" via MQTT with QoS 0
✔ Given client "A" subscribe to "foobar" via WEBSOCKET
Scenario: [MD_004] Receiving message from MQTT (QoS 0)
```

Figure 4.2. Integration Testing (WebSocket and MQTT)

From Figure 4.1 and Figure 4.2. it is concluded that the integration between two components (in this case are 2 nodes with different protocols) that can communicate from one component (node) to another (node).

4.2. Interoperability Testing.

This test is intended to determine the level of interoperability of the middleware designed. The middleware and node sensor devices are tested by using wireless networks whether the successful communication from sending data from the sensor node to the middleware is successful. If successful, how much success should be considered.

4.2.1. Testing of Hardware and Software

This test covers the needs of all hardware and software, especially in the middleware and sensor nodes, as well as a little discussion in the cloud. Figure 4.3 shows the middleware hardware that uses the Raspberry Pi. Next Figure 4.4 shows the sensor node device using NodeMCU

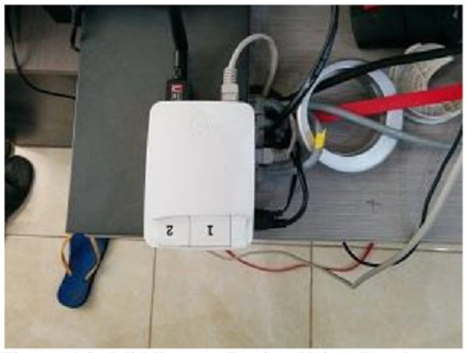


Figure 4.3. Middleware Device Using Raspberry Pi

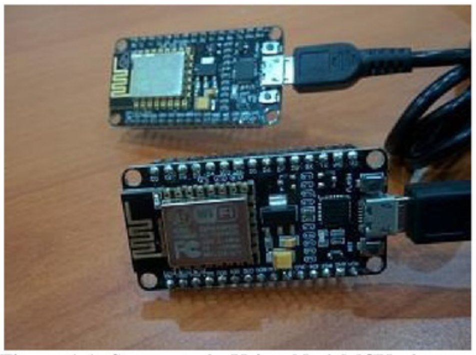


Figure 4.4. Sensor node Using NodeMCU along with the Sensor

Physically and electronically, the hardware can operate properly, meeting the expected functional objectives.

4.2.2. Success Rate of Delivery on Wireless Networks

Testing the success of data transmission between sensor nodes and middleware on wireless networks using wireless devices is indicated by packet loss rates and delay levels. The measurement results are shown in Table 4.1, where "Expected" is the amount of data sent at the sender, and "Actual" is the amount of data received at the recipient. While the results of measuring the delay in delivery are shown in Table 4.2. Measurement delay is based on the percentage of packet loss that is different. This is a representation of the state of wireless networks where packet loss is more likely to occur than using a wired network.

Table 4.1. Measurement of Packet Loss from the Sensor node to Middleware

CoAP				WebSocket				MQTT			
Expected	Actual	Success rate	Pakcet loss rate	Expected	Actual	Success rate	Pakcet loss rate	Expected	Actual	Success rate	Pakcet loss rate
120	104	86.67%	13.33%	120	101	84.17%	15.83%	120	116	96.67%	3.33%
120	109	90.83%	9.17%	120	109	90.83%	9.17%	120	118	98.33%	1.67%
120	114	95.00%	5.00%	120	100	83.33%	16.67%	120	106	88.33%	11.67%
120	**109**	**90.83%**	**9.17%**	**120**	**103**	**86.11%**	**13.89%**	**120**	**113**	**94.44%**	**5.56%**

Table 4.2. Delay Measurement from Sensor to Middleware

Packet Loss	Rata-Rata Delay		
	CoAP	WebSocket	MQTT
0%	0.00694422	0.00142248	0.00395457
5%	0.21076168	0.00136770	0.11953973
10%	0.37334771	0.00135949	0.34241269
15%	0.62471699	0.00141067	0.46357384
20%	0.83159735	0.00136134	0.66171421
25%	1.06828006	0.00173441	0.77042744

From Table 4.2. a graph can be created as shown in Figure 4.5.

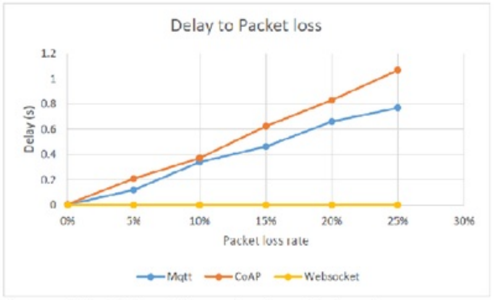
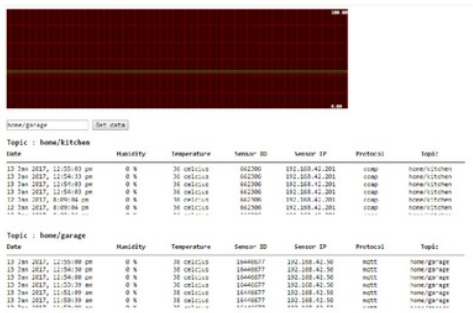


Figure 4.5. Delay Chart Against Packet Loss on Measurement
Sensor node to Middleware

From the results of measurement of packet loss, the MQTT protocol packeloss is the smallest compared to the other two protocols. This happens probably because the size of the packet data formed by MQTT is smaller, so the success of reading the data packet is greater.
From the results of delay measurements on percentage of packet loss, WebSocket shows a small delay compared to the others. This is probably due to the delay measurement process starting when "immediately" the packet data is sent until it is processed by the recipient. "Immediately" sent this data packet to WebSocket faster than others.
The data sent by the sensor is proven to be compatible with the data received and stored by the application, including data that has been

received by the application to be displayed and stored in the mongoDB database. The data displayed on the web-app is seen and compared to the data sent by the sensor. The results of these observations are shown in Figure 4.6.



Topic : home/kitchen

Date	Humidity	Temperature	Sensor ID	Sensor IP	Protocol	Topic
13 Jan 2017, 12:55:03 pm	0 %	36 celcius	662306	192.168.42.201	coap	home/kitchen
13 Jan 2017, 12:54:33 pm	0 %	36 celcius	662306	192.168.42.201	coap	home/kitchen
13 Jan 2017, 12:54:03 pm	0 %	36 celcius	662306	192.168.42.201	coap	home/kitchen
13 Jan 2017, 12:54:03 pm	0 %	36 celcius	662306	192.168.42.201	coap	home/kitchen
12 Jan 2017, 8:09:04 pm	0 %	36 celcius	662306	192.168.42.201	coap	home/kitchen
12 Jan 2017, 8:09:04 pm	0 %	36 celcius	662306	192.168.42.201	coap	home/kitchen

Topic : home/garage

Date	Humidity	Temperature	Sensor ID	Sensor IP	Protocol	Topic
13 Jan 2017, 12:55:00 pm	0 %	38 celcius	16446677	192.168.42.50	mqtt	home/garage
13 Jan 2017, 12:54:30 pm	0 %	38 celcius	16446677	192.168.42.50	mqtt	home/garage
13 Jan 2017, 12:54:00 pm	0 %	38 celcius	16446677	192.168.42.50	mqtt	home/garage
13 Jan 2017, 11:53:39 am	0 %	36 celcius	16446677	192.168.42.50	mqtt	home/garage
13 Jan 2017, 11:51:09 am	0 %	38 celcius	16446677	192.168.42.50	mqtt	home/garage
13 Jan 2017, 11:50:39 am	0 %	38 celcius	16446677	192.168.42.50	mqtt	home/garage

Figure 4.6. Data Display in Web Applications

5. Conclusion

Thus some conclusions can be taken from the results of the design and testing that have been done:
1. As planned, that this middleware is actor-based, that is, middleware can provide "decisions locally", which can serve as a bridge between protocol differences from the cloud and sensor nodes.
2. Communication between the CoAP, MQTT and Websocket protocols in the middleware can be achieved by creating a gateway for each protocol, then connecting with a broker.
3. The use of the Websocket protocol produces a lower delay value than the MQTT and CoAP protocols.

References

Anwari, H. 2017. *Pengembangan IoT Middleware Sebagai Gateway untuk Protokol CoAP, MQTT, dan Websocket*. Skripsi Fakultas Ilmu Komputer Universitas Brawijaya.

Atzori, L; Iera, A; Morabito, G. 2010. *The Internet of Things: A survey*. Computer Networks 54.15 (2010): 2787-2805.

Chaqfeh, M A.; Mohamed, N. 2012. *Challenges in Middleware Solutions for the Internet of Things*. Collaboration Technologies and Systems (CTS), 2012 International Conferenceon (2012): 21-25.

Fersi, G. 2015. *Middleware for Internet of Things: a study*. International Conferenceon Distributed Computing in Sensor System (2015): 230-235.

HiveMQ. 2015. *MQTT Essential Part 2 : Publish & Subscribe*. Dipetik February 28, 2016, dari HiveMQ: http://www.hivemq.com/blog/mqtt-essential-part2-publish-subscribe

IETF, I. E. 2016. The Constrained Application Protocol (CoAP) RFC7252. Internet Engineering Task Force.

Mahmoud, Q. 2005. *Middleware for Communications*. Wiley; June 2005.

Mohamed, M. A. 2012. *Challenges in Middleware Solutions for the Internet of Things*. 2012 International Conference on Collaboration Technologies and Systems (CTS), 21-26. doi:10.1109/CTS.2012.6261022.

Ngu, A H.; Gutierrez, M; Metsis, V; Nepal, S; Sheng, Q Z. 2017. *IoT Middleware: A Survey on Issues and Enabling Technologies*. IEEE Internet of Thnigs Journal 4.1 (2017): 1-20.

OASIS. 2014. MQTT Version 3.1.1.

Rahmani, A-M.; Thanigaivelan, N K.; Gia, T N.; Tenhunen, H.. 2015. *Smart e-healthgateway: Bringing intelligenceto internet-of-things based ubiquitous health care systems*.12th Annual IEEE Consumer Communications and Networking Conference (CCNC) (2015): 826-834.

JITECS ID 101

ORIGINALITY REPORT

7%	5%	7%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	link.springer.com Internet Source	1%
2	Abubakar Sadiq Sani, Dong Yuan, Jiong Jin, Longxiang Gao, Shui Yu, Zhao Yang Dong. "Cyber security framework for Internet of Things-based Energy Internet", Future Generation Computer Systems, 2018 Publication	1%
3	Goshi Sato, Yoshitaka Shibata, Noriki Uchida. "Chapter 48 Platform System Based on LoRa Mesh Network Technology", Springer Nature, 2019 Publication	1%
4	techpublications.org Internet Source	1%
5	Adeniyi Onasanya, Maher Elshakankiri. "Smart integrated IoT healthcare system for cancer care", Wireless Networks, 2019 Publication	1%
	www.joules.de	

6	Internet Source	1%
7	peerj.com Internet Source	<1%
8	Ipsita Koley, Tuhina Samanta. "Mobile sink based data collection for energy efficient coordination in wireless sensor network using cooperative game model", Telecommunication Systems, 2018 Publication	<1%
9	"Enterprise Interoperability VIII", Springer Science and Business Media LLC, 2019 Publication	<1%
10	Chih-Min Chao, , and Yi-Wei Lee. "A Quorum-Based Energy-Saving MAC Protocol Design for Wireless Sensor Networks", IEEE Transactions on Vehicular Technology, 2010. Publication	<1%
11	www.skb.se Internet Source	<1%
12	repository.up.ac.za Internet Source	<1%
13	"Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014", Springer Nature, 2015	<1%

Publication

14	Michael Walker, Douglas C. Schmidt, Jules White. "chapter 12 An Elastic Platform for Large-scale Assessment of Software Assignments for MOOCs (EPLASAM)", IGI Global, 2016 Publication	<1%
15	"Advances in Mobile Cloud Computing and Big Data in the 5G Era", Springer Nature, 2017 Publication	<1%
16	Soma Bandyopadhyay. "A Survey of Middleware for Internet of Things", Communications in Computer and Information Science, 2011 Publication	<1%
17	"A Practical Evaluation of a High-Security Energy-Efficient Gateway for IoT Fog Computing Applications", Sensors, 2017 Publication	<1%
18	Chaqfeh, Moumena A., and Nader Mohamed. "Challenges in middleware solutions for the internet of things", 2012 International Conference on Collaboration Technologies and Systems (CTS), 2012. Publication	<1%
19	Modeling and Optimization in Science and Technologies, 2016.	<1%

Publication

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		