An Internet-of-Things (IoT) system development and implementation for Automatic Weather Station (AWS) of BMKG based on MQTT Protocol

Ariffudin[1] , Ibnu Sofwan Lukito[2]

Center for Instrumentation, Calibration and engineering of
Meteorological Climatological and Geophysical Agency of Indonesia (BMKG)
E-mail: ariffudin@bmkg.go.id; ariffu@gmail.com

Abstract – With increasing appliances at Automatic Weather Station (AWS) observing system, the necessity to accommodate hundreds and thousands of sensor successfull automation of great prominence in the field of M2M communication FTP / HTTP protocol was well known for remote monitoring analysis of data form large number of sensing elements. Currently, BMKG (Meteorological, Climatological and Geophysical Agency) has operated AWS equipment about 430 site, each with 7 sensor parameter using FTP protocol. System using FTP / HTTP consume more power, had lesser efficiency of transmission and could not utilize system bandwidth efficiently. Traditional web communication technologies, such as FTP and HTTP, provide a uni-directional link and request/response message exchange model. The solution can be troublesome in web-based applications involving a large number of different interconnected device, such as in the IoT. That is an indeed very trending topic today and it is estimated that by 2020, almost 50 billion "things" will be connected to the internet.

The MQTT is one of the IoT protocol capable of handling sensor traffic under low bandwidth and constrained network conditions are extensively used to improve automated system. MQTT uses the priciple of publish – subsribe to communicating, is used because power saving and lightwight messaging protocol.

This paper presents a holistic conceptual approach of an Internet-of-Things (IoT) system development and implementation to enhance AWS of BMKG efficiently. The concept focuses on smart logger implemented using MQTT protocol for AWS applications has been proposed to solve above mentioned problem. The proposed logger is ARM platform for the implementation of MQTT client, for internet publishing and set of sensors for obtaining real time data.

***Keywords:* AWS, IoT, MQTT, FTP, BMKG, weather sensor.**

I. INTRODUCTION

Internet of Things (IoT) is being widely discussed. It is a topic of worldwide interest. In IoT, a large number of tiny data block from devices, such as various sensors of Automatic Weather Station (AWS), are transferred cross networks.

A total of approximately 430 automatic weather stations are run by the Meteorological Climatological and Geophysical Agency of Indonesia BMKG (Badan Meteorologi, Klimatologi dan Geofisika), including Local Meteorological Offices. Located nationwide, these stations routinely observe various meteorological elements such as surface pressure, temperature, humidity, wind direction/speed, precipitation, and solar radiation.

Currently, Internet access requires application protocols over TCP/IP or UDP/IP. One of the application protocols is Hyper Text Transfer Protocol (HTTP), which has been standardized in IETF, and has been applied for general communication over Internet. However, when HTTP is applied to communication in IoT, in which a huge number of tiny data blocks are transferred, protocol overhead and resulting performance degradation are a serious problem. Moreover, IP addressing depends on physical location, which causes the problem of complexity of network control. To solve these problems we need adopting the MQTT protocol for the IoT.

In these architectures, MQ Telemetry Transport (MQTT) is one of the protocols, suited for this scenario in which many devices have to exchange data between themselves in near real time through the Internet and we need to consume the least possible network bandwidth. MQTT reduces protocol overheads and provides high efficiency communication for IoT. It also invokes "Name based routing," and mitigates IP address based routing for IoT traffic flows.

II. HTTP AND FTP PROTOCOL

Internet network is built to communicate via HTTP (Hyper Text Transfer Protocol). Various data, from images to texts, are sent over internet every day. HTTP is as a primary protocol interface to move a wide range of data quickly, easily, and stable from server to user devices such as browser. HTTP is built on TCP. HTTP ensures that data transmitted from one device to another will not corrupt so that the integrity of data transmitted is assured. HTTP is an open communication protocol that can be read by any devices that have been developed for HTTP protocol as browser or smartphone through browser application. An HTTP transaction consists of two parts: request command (request) sent from client to server, and response command (response) sent from server to client. The process of response and request is submitted using a data block with specific format known as HTTP Message. The messages are sent by HTTP which moves in one direction.

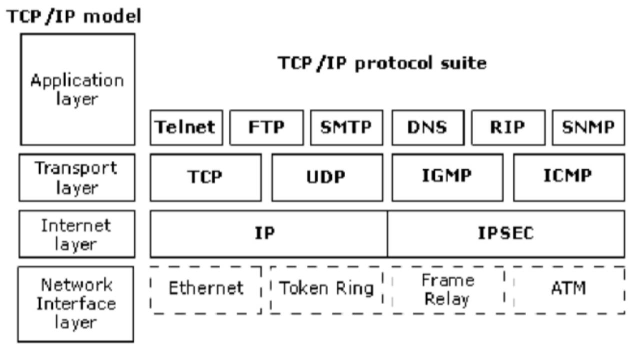


Fig. 1. TCP-IP Protocol suite

At this time, the BMKG monitoring system used is still using a data transmission system with the ftp protocol , fig 2

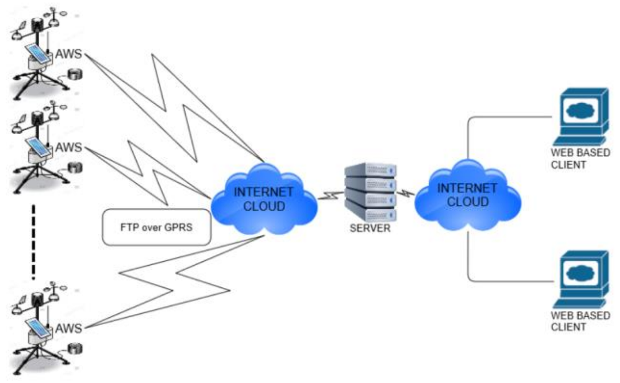


Fig. 2. Diagram of AWS with FTP protocol

III. MQTT PROTOCOL

MQTT (MQ Telemetry Transport) is one of the protocols supported by the IBM Message Broker products as a communicating data to and from the Broker . The protocol was designed specifically for remote telemetry applications, with three specific design goals: [1] It should offer a once-and-once-only assured delivery mode to enable a message to be reliably transferred all the way from a remote sensor to a back-end application.[2] The protocol should be as lightweight as possible across the "wire" (or other communication medium) most remote telemetry is done over low bandwidth, high cost networks, and so minimizing the overhead of each message is highly desirable. [3] The protocol should be very easy to implement on embedded devices such as sensors and gateways.

MQTT (Message Queuing Telemetry Transport) protocol is protocol specifically designed for "machine to machine" communication. MQTT protocol runs over TCP / IP and has a data packet size with low overhead minimum (> 2 bytes) so that consumption of the power supply is also small enough. This protocol is a data-agnostic protocol that can transmit data in various forms such as binary data, text, XML, or JSON and this protocol uses a publish/subscribe model rather than a client-server model.

Middleware is defined as the software which provides a messaging fabric to link applications and systems together. The alternative to not using a middleware system is that the application writer has to deal with the mechanics of getting messages from A to B, dealing with connection failures, network outages, duplicate messages etc. The use of middleware helps the application writer to communicate messages from one system to another system in remote locations. The IBM Web sphere MQ is one such messaging middleware that allows collaborating applications to intercommunicate via a central hub, known as a Message Broker [7]. Therefore data producers can produce desired set of data and just set up a MQTT publish to the Broker. On the other hand a Subscriber can subscribe to the published topic and extract data for actuation or remote monitoring.

A. A Pub / Sub in cloud based IoT

Today, several prominent academic and commercial IoT platforms share a cloud centric architecture similar to the one depicted in Figure 3. Devices are connected to gateways, which forward sensor data to a cloud tier with a message broker. Additional services, e.g. data storage, analysis, or aggregation, and user facing applications connect to the broker to get access to the data.

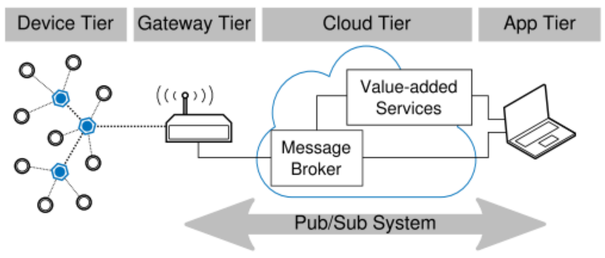


Fig. 3. General architecture of a cloud centic IoT Platform

We see a trend to use a message broker using the publish/subscribe pattern to distribute the data to multiple interested applications [4]. A publish/subscribe system is a message-oriented middleware (MoM) [5] providing distributed, asynchronous, loosely coupled communication between message producers and message consumers. A pub/sub middleware offers three main types of decoupling [6] which makes it particularly suitable for large-scale IoT deployments: 1) Message producers (publishers) and consumers (subscribers) are decoupled in time, i.e. they do not have to be connected at the same time; 2) Messages are not explicitly addressed to a specific consumer but to a symbolic address (channel, topic); 3) Messaging is asynchronous, non-blocking.

A core building block of pub/sub systems is the matching between publishers and subscribers, that may be based on different types of filtering, mostly topic or content. The filtering is usually done by multiple dedicated message brokers. In the topic based scheme, the symbolic channel addresses are topics, usually in the form of strings, i.e. producers publish to and consumers subscribe to topics. Messages are only delivered to matching subscribers. Topics may be organized hierarchically, i.e. a topic may be a subtopic of another topic. Subscriptions on a parent topic will then usually also match all subtopics. Topic based filtering is a static scheme offering only limited expressiveness. In contrast, in the content based scheme, subscribers are not statically matched based on topics, but on the content of individual messages, e.g. if a value reaches a certain threshold predefined by the subscriber.

A similar approach dealing with large scale sensor data is stream processing. In contrast to the message-based pub/sub, stream processing applications act as complex stateful continuous queries on input streams of data generating streams of results. The focus of stream processing frameworks lies in the transformation of the input stream, whereas pub/sub focuses on the distribution of data. While we see stream processing as a promising approach to process and analyze large data streams, we argue that stream processing alone will not enable the common use of sensor hardware across multiple applications, which is one of the key properties of the IoT vision. However, we envision value-added services that use stream processing approaches on the data provided by pub/sub systems.

This section outlines the specific IoT requirements that influence the selection of a suitable pub/sub middleware. We start with the functional requirements, that are derived from generic IoT applications :

1. Messaging Pattern: All use cases require the monitoring of sensor readings. We already motivated the use of the pub/sub pattern, where symbolic addresses are used to match producer and consumer, which has to be supported. It should additionally be possible to address and contact a particular device, e.g. automatic weather station monitoring system. That means a point to point messaging pattern should also be supported.

2. Filtering: Interested parties usually want to receive only a subset of all information, e.g. weather sensors in the same province. The filtering capabilities of the middleware determine the expressiveness of the subscriptions a client application can issue. A topic-based approach is suitable for basic subscriptions to

certain physical or virtual sensors. A hierarchical topic structure enables monitoring sensor sets. However, it often makes more sense to be informed on certain events, e.g. when a sensor reading reaches a threshold. This requires a content based pub/sub pattern or additional complex event processing. While a topic-based filtering is mandatory for cloud-based pub/sub systems, a content-based scheme is highly desirable.

3. QoS Semantics: While a loss of sensor data messages may be tolerable in some settings, others might require guaranteed delivery of messages. The middleware should therefore enable annotating subscriptions and messages with QoS requirements. Additionally, especially for sensors with low sampling rate, the system should provide subscribers with latest values while waiting for the next sensor reading.

4. Topology: In the context of cloud centric IoT, every pub/sub middleware must support a centralized topology, where a broker forwards the messages based on the requested filters. Please note that while we consider a single logical broker, this broker will be distributed across several physical or virtual machines.

5. Message format: Due to the heterogeneity of sensor hardware as illustrated in the selected use cases, it is challenging to foresee the exact format sensor data will be provided in. Pub/sub solutions must therefore be payload agnostic, i.e. make no assumptions about the payload. Further, they should support binary payloads, so that binary data serialization frameworks, such as protocol buffers can be used.

B. Broker

The publish-subscribe pattern requires a **broker**, also known as **server**. All the clients establish a connection with the broker. The client that sends a message through the broker is known as the **publisher**. The broker filters the incoming messages and distributes them to the clients that are interested in the type of received messages. The clients that register to the broker as interested in specific types of messages are known as **subscribers**. Hence, both publishers and subscribers establish a connection with the broker[8ok]. It is easy to understand how things work with a simple diagram. The following diagram shows one publisher and two subscribers connected to a broker:

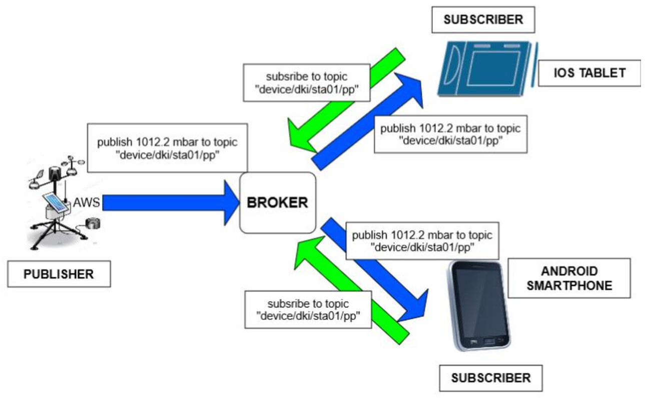

SUBSCRIBER
subsribe to topic "device/dki/sta01/pp"
IOS TABLET
publish 1012.2 mbar to topic "device/dki/sta01/pp"
publish 1012.2 mbar to topic "device/dki/sta01/pp"
AWS
BROKER
publish 1012.2 mbar to topic "device/dki/sta01/pp"
PUBLISHER
ANDROID SMARTPHONE
subsribe to topic "device/dki/sta01/pp"
SUBSCRIBER

Fig. 4. Diagram pub / sub mqtt

Publishers and subscribers are decoupled in space because they don't know each other. Publishers and subscribers don't have to run at the same time. The publisher can publish a message and the subscriber can receive it later. In addition, the publish operation isn't synchronized with the receive operation. A publisher requests the broker to publish a message and the different clients that have subscribed to the appropriate topic can receive the message at different times. The publisher can send messages as an asynchronous operation to avoid being blocked until the clients receive the messages. However, it is also possible to send a message to the broker as a synchronous operation with the broker and to continue the execution only after the operation was successful. In most cases, we will want to take advantage of asynchronous operations.
A publisher that requires sending a message to hundreds of clients can do it with a single publish operation to a broker. The broker is responsible for sending the published message to all the clients that have subscribed to the appropriate topic. Because publishers and subscribers are decoupled, the publisher doesn't know whether there is any subscriber that is going to listen to the messages it is going to send. Hence, sometimes it is necessary to make the subscriber become a publisher too and to publish a message indicating that it has received and processed a message. The specific requirements depend on the kind of solution we are building. MQTT offers many features that make our lives easier in many of the scenarios we have been analyzing.

IV. MQTT IMPLEMENTATION FOR AWS SYSTEM

The AWS system proposed in the paper is based on IoT technologies and consists of three important parts: MQTT Client Publisher, Server or Broker and MQTT Client Subscriber. The overall system is depicted as shown in Figure 1. The MQTT Client Publisher is a weather station with MQTT protocol implemented in Publish mode using Embedded microcontroller based software platform. The Publisher is capable to extracting sensor data from single or multiple sources and publishing it through topics on to a server hosted as part of public domain or private secure servers. Private servers offer a better security guarantee than public hosted servers. Server is a storage and computationally equipped hardware with support software. For the publisher to be able to communicate with the server efficiently a Broker MQTT
must be implemented on the server with the communication port specified to the Publisher. The MQTT Client Subscriber is device with MQTT Client implemented in Subscribe mode. The Subscriber is capable of accessing the data on the server through a subscription to particular topic.

A. Weather Station based on IoT using MQTT Protocol

An automatic weather station (AWS) is an automated version of the traditional weather station, either to save human labour or to enable measurements from remote areas. An AWS will typically consist of a weather-proof enclosure containing the data logger, rechargeable battery, telemetry and the meteorological sensors with an attached solar panel and mounted upon a mast. The specific configuration may vary due to the purpose of the system.

No.	**Sensor**	**Parameter**	**Unit**
1.	Wind speed	WS ave_10mnt	m/s
		WS_max_10 mnt	m/s
2.	Wind direction	WD_ave_10 mnt	degree
3.	Temperature	Temp_av_10mnt	ºC
		Temp_max_10mnt	ºC
		Temp_min_10mnt	ºC
4.	Humidity	Humidity_ave_10mnt	%RH
5.	Pressure	Pressure_ave_10mnt	mbar
6.	Rain	Rain_acc 24 hours	mm
7.	Solar Rad	SR_ave_10mnt	W/m2
		SR_max_10mnt	W/m2

tabel. 1. Format of data sent to the server

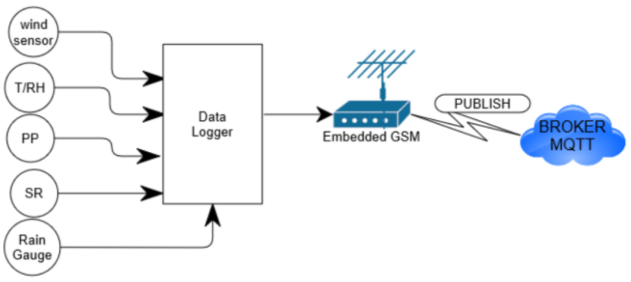


Fig. 5. Diagram weather station

1. Data logger
The data logger is the heart of the weather station and placed in enclosure along with other equipment such as battery, regulator, pressure sensor, and embedded GSM communication.
2. Wind sensor, wind monitor RM Young
3. Temperature and humidity sensor HMP155 Vaisala
4. Pressure sensor PTB210 Vaisala
5. Rain sensor with tipping bucket
6. Solar radiation sensor form kipp and zonen
7. Embedded modem communication form logicio
Logicio embedded MX2 turbo has been designed for abroad variety of advanced telemetry applications and is made according to the highest technical standard for professional and industries use. Technical highlights :

- Based on M2M platform
- Support MQTT protocol
- Huge standard API with function
- 3G GSM engine
- I/O analog and digital
- IDE development tool with device emulator

B. Mosquitto broker

Mosquitto provides standards compliant server and client implementations of the MQTT messaging protocol. MQTT uses a publish/subscribe model, has low network overhead and can be implemented on low power devices such microcontrollers that might be used in remote Internet of Things sensors[9]. As such, Mosquitto is intended for use in all situations where there is a need for lightweight messaging, particularly on constrained devices with limited resources.

The Mosquitto project is a member of the Eclipse Foundation
There are three parts to the project.
• The main mosquitto server
• The mosquitto_pub and mosquitto_sub client utilities that are one method of communicating
• An MQTT client library.
Mosquitto supports other research activities as a useful block for building larger systems and has been used to evaluate MQTT for use in Smart City Services, and in the development of an environmental monitoring system.

1. Filtering with the topic

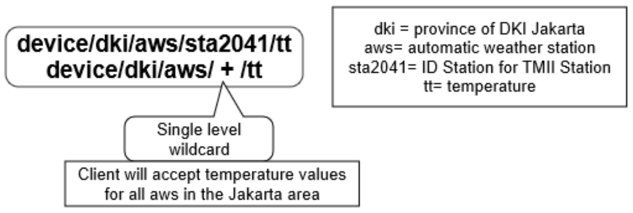
The message topic is a simple string that could have more hierarchy levels separated with a slash. This means if we want to publish the temperature of the weather station di spesific province, we can use a topic like this: device/dki/sta2041/tt. The hierarchy becomes really relevant when we subscribe our devices to more topics.

device/province/aws_type/id_station/sensor
province = provinces in the Indonesian territory (34)
aws_type = aws / arg / aaws
id_station = observation station code
sensor =

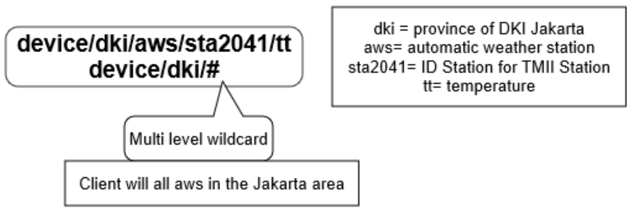
- time = time stamp for observation (UTC)
- rr=rain (accumulation)
- ws=wind speed (average)
- ws_max= wind speed maximum
- wd=wind direction (average)
- tt = temperatur (average)
- tt_max=temperature maximum
- tt_min=temperature minimum
- sr=solar radiation (average)
- sr_max= Solar Radiation Maximum

If we suppose that we have another sensor we will have a publisher that sends messages to the topic like this: device/dki/sta2041/tt. In this case, if another device needs to subscribe to all temperature messages regardless of the location, we can subscribe it to multiple topics using a wildcard. MQTT provides two different wildcards:

• The + operator is a single-level wildcard that allows arbitrary values for one hierarchy. With the preceding example, the device/dki/aws/+/tt topic will subscribe to all temperature updates, all station in Jakarta province.


device/dki/aws/sta2041/tt
device/dki/aws/ + /tt
dki = province of DKI Jakarta
aws= automatic weather station
sta2041= ID Station for TMII Station
tt= temperature
Single level wildcard
Client will accept temperature values for all aws in the Jakarta area

• The # operator is a multilevel wildcard that allows subscribing to all the underlying levels. With the preceding example, the device/dki/# topic will subscribe to any topic that begins with the /dki (jakarta province) string.


device/dki/aws/sta2041/tt
device/dki/#
dki = province of DKI Jakarta
aws= automatic weather station
sta2041= ID Station for TMII Station
tt= temperature
Multi level wildcard
Client will all aws in the Jakarta area

2. Security

MQTT allows to send a username and password for authenticating the client and also authorization[10]. MQTT has a security problem in terms of privacy, to ensure client's identity which access MQTT protocol required an authentication and authorization mechanisms, that can be achieved by applying Access Control List (ACLs) to the broker that will govern the rights of client to access certain topic on system, such as publish/subscribe.

3. Quality of Service level

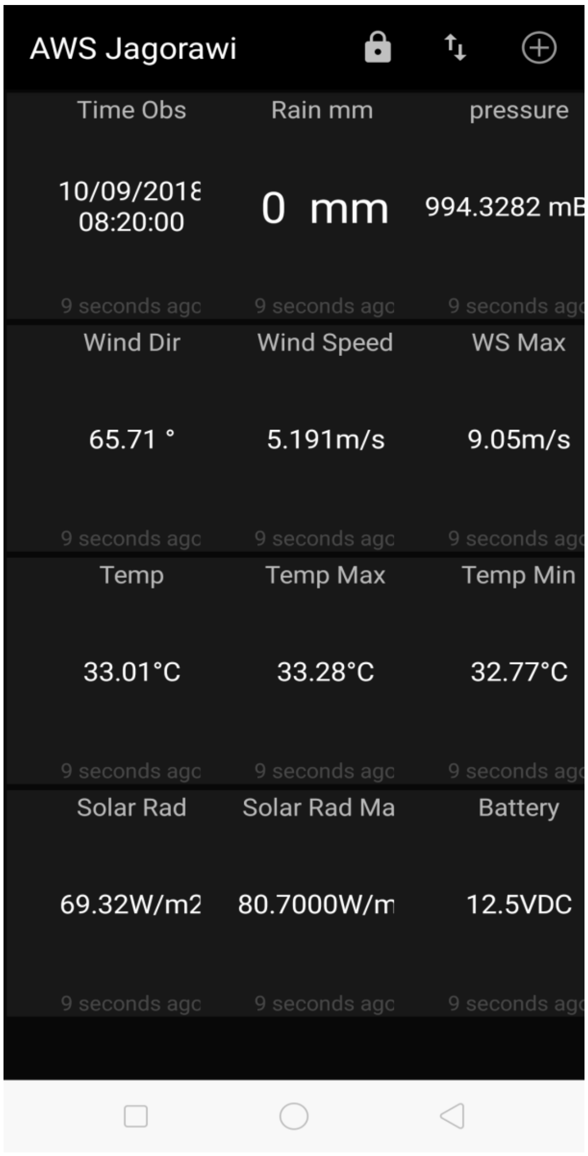
During the trial period using QOS 0, level 0 would send the message and also called as send it and never acknowledge. It is just a one-time send without confirmation about the message reaching the destination. It is more suited to situations where the importance is low [11]

C. MQTT Client (Subscriber)

A subscriber is usually a node or device which is interested in receiving data. Data from other nodes or devices is received by the subscribed device or node. The only condition here is that the subscriber needs to be subscribed to that particular topic on which the publisher is sending data. The subscriber gets the message after it is published by the publisher.

An IOT platform is an hardware and software system for managing IOT devices and collecting, storing, visualising and analysing data from those devices. There are many IOT platforms on the market, and the functionality of these platforms varies enormously.

MQTT client for android smartphone such as MQTT dash is one of the suscriber. The screenshot



AWS Jagorawi

Time Obs	Rain mm	pressure
10/09/2018 08:20:00	0 mm	994.3282 mB
9 seconds ago	9 seconds ago	9 seconds ago
Wind Dir	Wind Speed	WS Max
65.71 °	5.191m/s	9.05m/s
9 seconds ago	9 seconds ago	9 seconds ago
Temp	Temp Max	Temp Min
33.01°C	33.28°C	32.77°C
9 seconds ago	9 seconds ago	9 seconds ago
Solar Rad	Solar Rad Ma	Battery
69.32W/m2	80.7000W/m	12.5VDC
9 seconds ago	9 seconds ago	9 seconds ago

V. CONCLUSION

This study has implemented the use of MQTT protocol to build weather station system application with a embedded modem communication interface, which is android and web-based. The test result indicates that MQTT protocol has the ability of transfer data faster than HTTP protocol, which can transfer amount of data 6 times of HTTP capability. MQTT usage can be an option for the hardware data acquisition real-time application based on the Internet of Things.

REFERENCES

[1] Bandyopadhyay, S.; Bhattacharyya, A., "Lightweight Internet protocols for web enablement of sensors using constrained gateway devices," Computing, Networking and Communications (ICNC), 2013 International Conference on , vol., no., pp.334,340, 28-31 Jan. 2013.

[2] Colitti, Walter, Kris Steenhaut, and Niccolò De Caro. "Integrating wireless sensor networks with the web." Extending the Internet to Low power and Lossy Networks (IP+ SN 2011) (2011).

[3] Ming Wang; Guiqing Zhang; Chenghui Zhang; Jianbin Zhang; Chengdong Li, "An IoT-based appliance control system for smart homes," Intelligent Control and Information Processing (ICICIP), 2013 Fourth International Conference on , vol., no., pp.744,747, 9-11 June 2013

[4] Menzel, T., Karowski, N., Happ, D., Handziski, V., Wolisz, A.: Social sensor cloud: An architecture meeting cloud-centric iot platform requirements (2014). 9th KuVS NGSDP Expert Talk on Next Generation Service Delivery Platforms

[5] Curry, E.: Message-oriented middleware. In: Q.H. Mahmoud (ed.) Middleware for Communications, chap. 1, pp. 1–28. John Wiley & Sons (2005)

[6] Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The many faces of publish/subscribe. ACM Computing Surveys (CSUR) 35(2), 114–131 (2003)

[7] Ullas BS, Anush S, Roopa J, Govinda Raju M, “Machine to Machine communication for smart system using MQTT” IJAREEIE, March 2014.

[8] Gaston C Hillar, “MQTT Essential – a lightweight IoT Protocol” April 2017 packt publishing, Birmingham

[9] Roger Alan Light, “Mosquitto : server and client implementation of the MQTT protocol”, may 2017, paper DOI, Uniersity of Nottinghem.

[10] Sumit Pal, “ study and implementation of environment monitoring system based in MQTT”, EESRJ, IIETA, march 2017,

[11] Priyanka Thota, “Implementation and analysis of communication protocol in internet of things”, may 2017, UNLV, University Libraries, Nevada, Las Vegas

[12] M. Author, “Paper submitted to publication in a periodical,” *Sci. Bull. Politehnica Univ. Timisoara Trans. Autom. Control Comput. Sci.*, submitted for publication.