Over The Air Update Firmware pada Perangkat IoT Dengan Protokol MQTT

Luqman Hakim[1], Wahyu Andhyka Kusuma[2], Mahar Faiqurahman[3], Supriyanto[4]
Universitas Muhammadiyah Malang
e-mail: [1]luqman.hakim@umm.ac.id, [2]wahyukusuma@umm.ac.id, [3]maharf@gmail.com,
[4]supriyanto4id@gmail.com

Diajukan: 5 November 2019; Direvisi: 13 Mei 2020; Diterima: 3 Juni 2020

Abstrak

Perangkat IoT yang diimplementasi pada banyak tempat dapat mengalami perubahan berupa update firmware. Update firmware pada perangkat IoT biasanya dilakukan dengan mengambil perangkat IoT, lalu menghubungkan ke komputer menggunakan komunikasi serial melalui kabel USB to micro USB, selanjutnya melakukan update firmware pada perangkat IoT dan mengembalikan perangkat IoT ke tempat. Jika sistem pada perangkat IoT sudah dapat berkomunikasi melalui antar muka jaringan, tidak perlu lagi menggunakan kabel USB to micro USB, karena bisa dimanfaatkan over the air update firmware menggunakan antar muka jaringan pada perangkat IoT. Over the air update firmware adalah memuat firmware hasil build dari Arduino IDE pada perangkat IoT menggunakan antar muka jaringan Wi-Fi, pada penelitian ini perangkat IoT menggunakan mikrokontroler ESP8266-12E. Untuk melakukan update firmware perangkat IoT digunakan protokol MQTT untuk menjembatani antara aplikasi berbasis website sebagai interface pengguna untuk publish file firmware ke perangkat IoT. Hasil dari implementasi aplikasi berbasis website untuk over the air update firmware pada perangkat IoT dengan protokol MQTT, dalam 10 kali pengujian pengiriman file firmware perangkat IoT menggunakan masing-masing QoS 0, QoS 1, dan QoS 2, didapatkan hasil QoS 2 lebih direkomendasikan untuk digunakan mengirim file firmware dengan keberhasilan update firmware QoS 0 = 50 %, QoS 1 = 70% dan QoS 2 = 80% dari 10 kali percobaan pengiriman file firmware pada perangkat IoT.

Kata kunci: *Firmware Update, QoS, IoT, MQTT.*

Abstract

IoT devices that are implemented in many places can experience changes in the form of firmware updates. Firmware update on an IoT device is usually done by taking an IoT device, then connecting to the computer using serial communication via a USB to micro USB cable, then updating the firmware on the IoT device and returning the IoT device to its place. If the system on the IoT device is able to communicate via a network interface, there is no need to use a USB to Micro USB cable, because it can be utilized over the air firmware update using the network interface on the IoT device. Over the air firmware update is loading firmware builds from Arduino Idea on an IoT device using a Wi-Fi network interface, in this study the IoT device uses an ES8266-12E microcontroller. Firmware update of the IoT device the MQTT protocol is used to bridge the website-based application as a user interfaces for PUBLISH firmware files to the IoT device. The results of the implementation of a website-based application for over the air firmware update on IoT devices with the MQTT protocol, in 10 times testing the sending of IoT device firmware files using each of QoS 0, QoS 1, and QoS 2, the results obtained QoS 2 are more recommended for use sending firmware files with successful firmware update QoS 0 = 50%, QoS 1 = 70% and QoS 2 = 80% of 10 attempts to test the firmware file on an IoT device.

Keywords*: Firmware Update, QoS, IoT, MQTT.*

1. Pendahuluan

Pada akhir 2013, ada 9,1 miliar unit perangkat *Internet of Things* (IoT) dengan konektivitas *Internet Protocol* dan berkomunikasi tanpa interaksi dengan manusia. *Internasional Data Corporation* (IDC) memperkirakan pertumbuhan IoT yang diterapkan tiap tahun mencapai 17.5% diperkirakan menjadi 28,1 miliar di tahun 2020 [1]. Bahkan Cisco mempunyai prediksi dua kali lipat lebih besar yaitu 50 miliar pada tahun 2020 [2]. Perangkat IoT yang digunakan di berbagai tempat sering dianggap sebagai sistem

yang tidak perlu mengubah *requirements* dan fungsinya. Namun, pada kenyataannya di mana perangkat IoT ini berjalan pasti akan berubah [3]. Perubahan ini meliputi perubahan *behavior*, parameter yang terkait komunikasi dengan sistem lain atau pengguna, memperbaiki kesalahan, bisa masalah keamanan, yang dilaporkan pengguna setelah perangkat IoT digunakan [3].

Berbagai perubahan perangkat IoT dapat dilakukan dengan mengganti *firmware*, untuk mengganti *firmware* pada perangkat IoT harus keluar mengambil perangkat IoT, menghubungkan ke komputer, melakukan *update*, dan mengembalikan perangkat IoT ke tempat. Namun, hal ini tidak dapat terus dilakukan bagi perusahaan yang memiliki perangkat IoT di berbagai tempat, seperti yang dilakukan *Chrysler* "merek mobil" pada tahun 2015 mereka dikritik karena mengirim perangkat *flashdisk* ke pelanggan untuk melakukan *update firmware*. Karena sangat rentan, *flashdisk* dapat diambil, dimodifikasi, dan dikirim kembali [4].

Jika sistem pada perangkat IoT sudah dapat berkomunikasi melalui antarmuka jaringan, hal ini bisa dimanfaatkan untuk menerapkan pembaruan *firmware* pada sistem IoT yang disebut dengan *Over The Air* (OTA) [3]. OTA dilakukan oleh *Tesla* pada tahun 2016 dengan mengirimkan pembaruan *firmware* pada mobil mereka dan konsumen dapat mengatur akan melakukan pembaruan pada saat mobil diparkir [4].

Over the air update adalah proses memuat *firmware* pada modul ESP "perangkat IoT" menggunakan koneksi jaringan Wi-Fi daripada menggunakan kabel *port serial* [5]. Secara umum istilah OTA adalah mekanisme penggunaan *wireless* untuk mengirim data, memperbarui paket untuk pembaruan *firmware* atau perangkat lunak ke perangkat *mobile*, sehingga pengguna tidak perlu pergi mengakses fisik perangkat untuk mengubah aplikasi, parameter, *firmware*, atau memperbarui *software* [6].

Over the air update pada perangkat IoT sudah ada di pasaran dalam produk merek Libelium, namun OTA hanya bisa dilakukan pada perangkat IoT Libelium melalui *File Tranfer Protocol* (FTP). Selain itu ada *particle.io* sama seperti Libelium hanya bisa digunakan untuk perangkat IoT yang mereka sediakan tidak bisa untuk perangkat lain. Selain dua perangkat berbayar Libelium dan *particle.io*, OTA di sediakan oleh Espressif melalui produk bernama ESP8266-12E, dapat melakukan *update firmware* melalui Arduino IDE, *web browser* , dan HTTP *Server* [5].

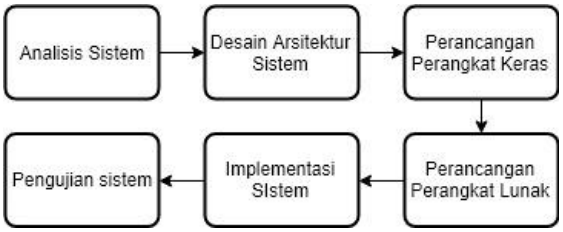
Over the air update dengan ESP8266-12E yang telah tersedia melalui Arduino IDE dan *web browser* memilik keterbatasan hanya bisa dilakukan dalam satu jaringan yang sama, dan HTTP *Server* bisa dilakukan dengan jaringan berbeda [5]. Namun pada penelitian pengujian *sensing* data suhu dan kelembapan pada penelitian sebelumnya menunjukkan protokol HTTP 6 kali lebih lambat melakukan transfer data daripada protokol MQTT dalam 60 detik. Dari 5 kali percobaan didapatkan rata-rata HTTP 934.4 data terkirim dan MQTT 6520.2 data terkirim [7].

Protokol MQTT adalah protokol pesan *publish/subcribe*, sangat sederhana, dan ringan, dirancang untuk perangkat yang terbatas oleh jaringan dengan *bandwidth* rendah, latensi tinggi, atau tidak dapat diandalkan. Prinsip desain protokol MQTT adalah untuk meminimalkan *bandwidth* jaringan dan kebutuhan sumber daya perangkat, sambil berusaha memastikan keandalan dan beberapa tingkat kepastian pengiriman data benar-benar terkirim *Quality of Service* (QoS) [8]. Contoh penggunaan protokol MQTT adalah Facebook Messenger [9] pada awal peluncurannya tahun 2011.

Berdasarkan latar belakang tersebut akan diimplementasikan protokol MQTT yang dapat melakukan *over the air update* pada perangkat IoT. *Over the air update* dengan protokol MQTT akan digunakan untuk melakukan *update firmware* dan *software* pada perangkat IoT, dari jaringan lokal atau jaringan internet, melalui media aplikasi berbasis *website*. Aplikasi berbasis *website* sebagai media *interface* pengguna untuk melakukan *update file firmware* dan melakukan *monitoring* hasil *upload* apakah berhasil atau tidak. Protokol MQTT digunakan sebagai media pengiriman *file firmware* hasil *build* dari *Arduino IDE* berupa *file* tipe bin, yang di-*publish* ke perangkat IoT yang telah melakukan s*ubscribe* pada suatu *topic*.

2. Metode Penelitian

Penerapan *over the air update firmware* dengan protokol MQTT dengan aplikasi berbasis *website* sebagai *interface* pengguna, dilakukan dengan beberapa tahapan, yaitu analisis sistem, desain arsitektur sistem, perancangan perangkat keras, perancangan perangkat lunak, implementasi sistem, dan pengujian sistem.



Gambar 1. Alur penelitian.

2.1. Analisis Sistem

Dari analisis masalah, maka dibuat *server* yang telah dikonfigurasi, untuk menjadi *web server*, menyimpan aplikasi berbasis web sebagai *interface* pengguna dan protokol MQTT berfungsi untuk jembatan komunikasi antara *node* perangkat IoT. Protokol MQTT adalah protokol komunikasi *subscribe* dan *publish*, di mana sebelum mengirim dan menerima pesan *client* harus terlebih dahulu terkoneksi dengan koneksi TCP ke *server broker* MQTT yang telah dikonfigurasi, selanjutnya *publisher* dan *subscriber* harus memiliki topik yang sama untuk saling komunikasi.

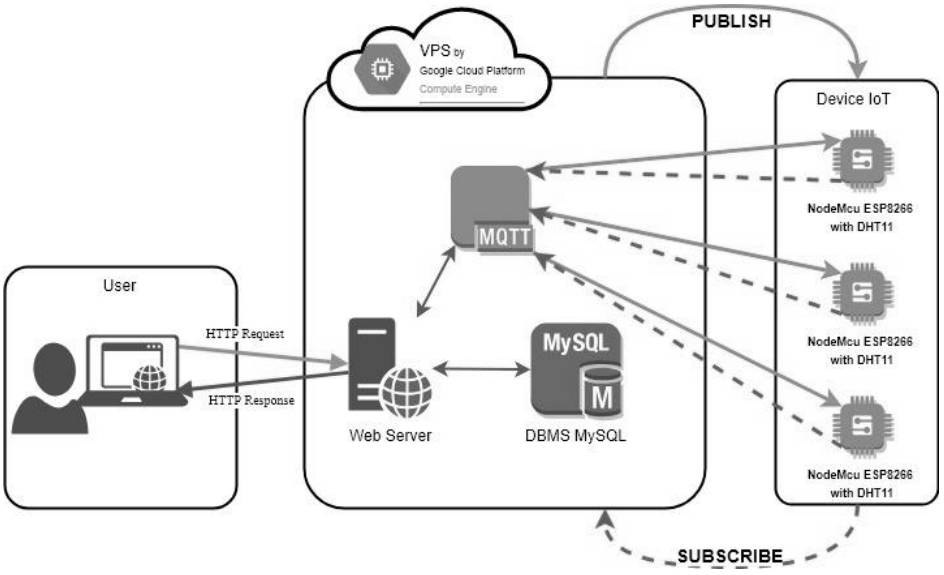
Publisher akan mengirim data ke *server broker* MQTT lalu *broker* akan mengirim ke *subscriber*. Identifikasi antara *publisher* dan *subscriber* harus memiliki topik yang sama agar pesan tersampaikan. Contohnya topik "rumah/dapur/suhu", jadi setiap *publisher* dan *subscriber* akan memiliki topik yang sama.

Over the air update firmware kepada perangkat IoT dilakukan dengan cara mengambil *file firmware* hasil *compile* dari Arduino IDE dengan ekstensi .bin, selanjutnya melakukan *upload* ke aplikasi berbasis *web* dan *publish* ke *broker* MQTT, lalu perangkat IoT melakukan *subscribe* dan menerima *file* ekstensi .bin selanjutnya melakukan *update*.

Aplikasi berbasis *web* dibuat dengan PHP Framework Codeigniter, pengguna harus *login* terlebih dulu, setelah *login* akan tampil pilihan untuk *upload file firmware* ekstensi .bin, *form publish* topik tujuan, dan *button publish*.

2.2. Desain Arsitektur Sistem

Arsitektur sistem penerapan protokol MQTT untuk *over the air update firmware*, menggunakan aplikasi berbasis *website* sebagai *interface* pengguna dengan rancangan topologi sebagai berikut.



Gambar 2. Arsitektur *system* protokol MQTT untuk OTA *update firmware* perangkat IoT.

Dari rancangan topologi Gambar 2, terdiri dari *Vitual Private Server* (VPS) yang dikonfigurasi dengan protokol MQTT, *web server* Apache2, dan DBMS MySQL. Dalam penelitian ini, perangkat IoT yang digunakan adalah modul NodeMCU ESP8266-12E sebagai sensor *node*. Untuk *sensing* data suhu dan

kelembapan menggunakan DHT 11. Sensor *node* tersebut akan menerima pembaruan *firmware* dan pengguna yang akan mengakses aplikasi berbasis web melalui browser.

Agar perangkat IoT dapat terhubung ke VPS, maka harus dihubungkan dengan Wi-Fi yang sudah terhubung ke internet. Begitu juga dengan aplikasi berbasis *website* yang akan diakses oleh pengguna melalui *browser*, harus dipastikan memiliki nama domain atau IP publik yang bisa diakses melalui browser.

Agar rancangan arsitektur sistem Gambar 2 dapat dilaksanakan sesuai dengan rancangan yang telah dibuat, maka dibutuhkan *hardware* dan *software* sebagai berikut;

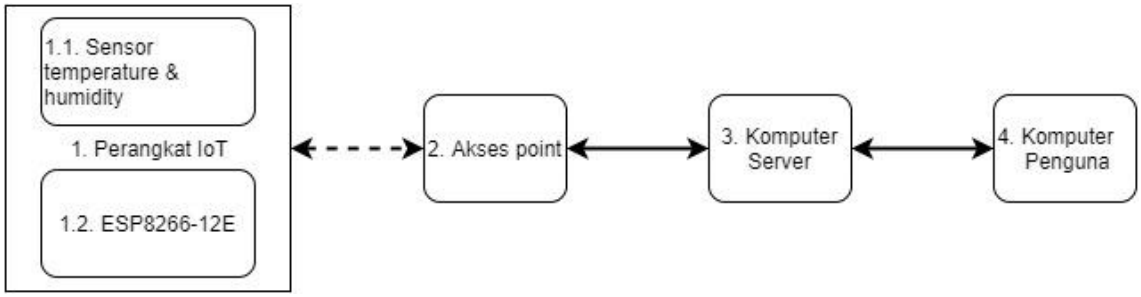
Tabel 1. *Software* dan *hardware* sistem OTA dengan MQTT.

Software	**Hardware**
Pengguna	Laptop
Windows 10	Kabel USB 2.0 to Micro USB
Browser	
PuTTy	
Atom IDE	
Server VPS	
LAMP (Linux, Apache, MySQL, PHP)	CPU 1 GHz
Linux Ubuntu Server 16.04	RAM 1 GB
Broker EMQTT 2.0	Harddisk 10 GB
Perangkat IoT	
Arduino IDE	NodeMCU ESP8266-12E
	Sensor DHT 11

Dari Tabel 1, *hardware* dan *software* terbagi menjadi 3 jenis yaitu untuk pengguna, *Virtual Private Server*, dan perangkat IoT.

2.3. Rancangan Perangkat Keras

Dari rancangan arsitektur, sistem *over the air update firmware* dengan protokol MQTT menggunakan aplikasi berbasis *website* sebagai *interface* pengguna. Rancangan *hardware* terdiri dari perangkat IoT, akses poin, komputer server, dan komputer pengguna seperti yang terlihat pada Gambar 3 di bawah ini.



Gambar 3. Rancangan *hardware* protokol MQTT untuk OTA *update firmware* perangkat IoT.

Keterangan:

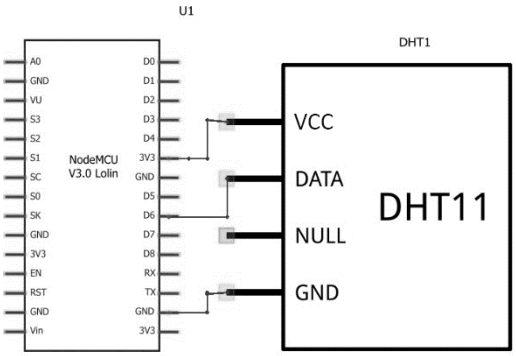
◄---► :Terhubung dengan jaringan *Wi-Fi*

◄——► :Terhubung ke jaringan internet

1. Perangkat IoT terdiri dari mikrokontroler dan sensor. Sensor *temperature* dan *humidity* menggunakan DHT 11 dan mikrokontroler menggunakan NodeMCU ESP8266-12E.
2. Akses poin yang sudah terhubung ke internet.
3. Komputer Server sebagai tempat meng-*install* web server Linux, Apache, MYSQL, PHP, dan meng-*install* EMQTT Broker.
4. Komputer pengguna untuk pengguna melakukan akses ke aplikasi berbasis *website* melalui browser untuk melakukan OTA dengan MQTT.

Perangkat IoT terdiri dari NodeMCU ESP8266-12E dan sensor DHT 11. NodeMCU ESP8266-12E digunakan untuk menerima *update firmware* dari aplikasi berbasis *website* yang dikirim oleh pengguna melalui protokol MQTT dan Sensor DHT 11 digunakan untuk *monitoring* apakah program pada *firmware* yang dikirim berfungsi atau tidak. Selain itu juga sensor digunakan untuk mengirim data temperatur dan *humidity* di sekitar perangkat IoT yang dikirim ke aplikasi berbasis *website*. Jadi akan dibuat dua *firmware*

yang telah terisi program *publish* data sensor, dan *subscribe* topik *firmware* yang akan digunakan untuk menerima *update* dari aplikasi berbasis *website*, dengan rancangan skematis sebagai berikut.

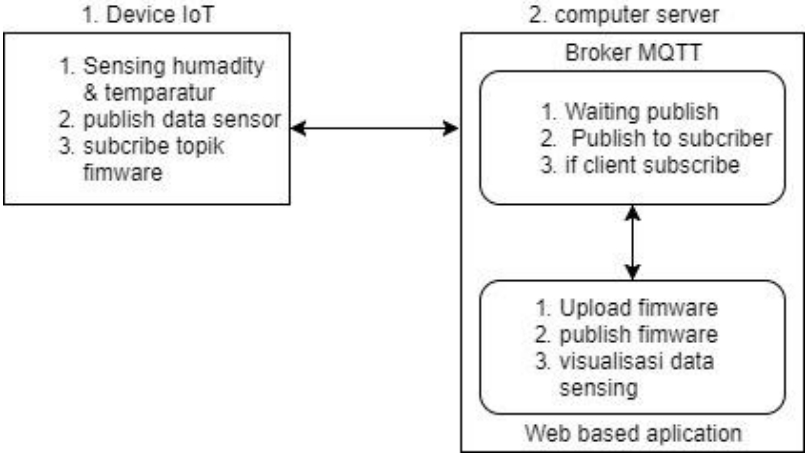


Gambar 4. Rangkaian skematis perangkat IoT.

Dari rangkaian skematis Gambar 4 DHT 11 memiliki 4 *pin* yaitu VCC, DATA, NULL, dan GND. *Pin* ini dihubungkan ke NodeMCU ESP8266-12E yaitu VCC dihubungkan ke 3V3, GND dihubungkan ke GND, dan DATA di hubungkan ke D6 pada NodeMCU ESP8266-12E.

2.4. Rancangan Perangkat Lunak

Perancangan perangkat lunak untuk setiap komponen pada sistem *over the air update firmware* dengan protokol MQTT, menggunakan aplikasi berbasis *website* yang akan diaplikasikan pada perangkat IoT dan server. Berikut adalah rancangan proses perangkat lunak pada Gambar 5.



Gambar 5. Perancangan proses perangkat lunak.

Keterangan:

1. Desain proses perangkat lunak pada perangkat IoT.
2. Desain proses perangkat lunak pada komputer *server*.

◄——► Jaringan internet.

Dari desain perancangan proses perangkat lunak pada Gambar 5, perangkat lunak dibagi menjadi 2 bagian perangkat lunak pada perangkat IoT dan pada server. Perangkat lunak pada perangkat IoT melakukan s*ensing* data sensor suhu dan kelembapan lalu melakukan *publish* data sensor dan melakukan *subscribe* topik *firmware* untuk menerima data *firmware* dari *publish* yang dilakukan aplikasi berbasis *website*. Pada *server broker* MQTT menunggu *publish* dari pengguna, jika ada yang melakukan *publish* dari pengguna maka *broker* akan melakukan *publish* kepada pengguna yang melakukan *subscribe* dan memiliki topik yang sama, pada aplikasi berbasis *website* melakukan *subscribe* data sensor dari perangkat IoT dan melakukan *upload firmware* lalu melakukan *publish file* .bin ke *broker* MQTT, lalu meneruskan ke perangkat IoT yang memiliki topik sama.

3. Hasil dan Pembahasan

Setelah rancangan *hardware* dan *software*, selanjutnya adalah dilakukan pengujian untuk melihat kemampuan *broker* MQTT untuk mengirim *file firmware* melalui aplikasi berbasis *website* ke perangkat IoT dengan tahapan pengujian sebagai berikut.

a. Uji coba pengiriman *file firmware* konstan 276 KB untuk mendapatkan tingkat kegagalan dan keberhasilan *update firmware* pada perangkat IoT
b. Uji coba apakah kegagalan dan keberhasilan *update firmware* pada perangkat IoT berpengaruh terhadap *delay*

Dari hasil pengujian pengiriman *file firmware* konstan 276 KB untuk melakukan *over the air update firmware* pada perangkat IoT menggunakan aplikasi berbasis *website*, dibuat analisis untuk digunakan sebagai rekomendasi QoS yang digunakan saat mengirim *file firmware* dengan dua sub bab analisis antara lain: Pengaruh Penggunaan QoS Terhadap Keberhasilan dan Kegagalan *Update Firmware* pada Perangkat IoT dan Pengaruh Keberhasilan dan Kegagalan *Update Firmware* pada Perangkat IoT, terhadap *Delay*.

3.1. Pengaruh Penggunaan QoS Terhadap Keberhasilan dan Kegagalan *Update Firmware* pada Perangkat IoT

Dari 10 kali pengiriman *file firmware* ke perangkat IoT dengan menggunakan setiap QoS 0, QoS 1, dan QoS 2 didapatkan hasil perangkat IoT berhasil di-*update* dan *restart* sukses dan gagal sebagai berikut.

Tabel 2. Perangkat IoT sukses *update firmware* dan *restart*.

QoS	*Updated Successfully*
QoS 0	50%
QoS 1	70%
QoS 2	80%

Dari Tabel 2 *firmware* yang berhasil di-*update* ke perangkat IoT, pada saat pengiriman dengan 10 kali percobaan pada QoS 0 = 50%, QOS 1 = 70%, dan QoS 2 = 80%. Hasil *update firmware* ke perangkat IoT tersebut terjadi karena *Quality of Service* setiap QoS 0, QoS 1, dan QoS 2 memiliki paket kontrol MQTT yang berbeda-beda, sehingga jika menggunakan QoS 0 tingkat keberhasilan *update firmware* 50% hanya melakukan *publish firmware* tanpa menerima kembali *acknowledge puback*. Jika menggunakan QoS 1 dengan tingkat keberhasilan *update firmware* 70 %, *publish firmware* akan mendapatkan balasan kembali berupa *puback*. Jika menggunakan QoS 2 dengan tingkat keberhasilan *update firmare* 80%, paket kontrol MQTT yang digunakan pada saat proses *publish firmware* memiliki *acknowledge* sebanyak tiga paket kontrol MQTT yaitu *pubrec*, *pubrel*, dan *pubcom*, yang memastikan *firmware* benar-benar terkirim.

3.2. Pengaruh Keberhasilan dan Kegagalan *Update Firmware* pada Perangkat IoT terhadap Delay

Pengujian performa mengirim *file firmware* konstan, hanya menghitung *delay* yang terjadi dalam sistem ketika mengirim *file firmware* antara aplikasi berbasis *website* saat *publish* dan saat perangkat IoT menerima kontrol paket MQTT berisi *file firmware*, namun tidak menghitung *delay* yang terjadi pada saat perangkat IoT menerima *file firmware* lalu melakukan *update* kemudian melakukan *restart* perangkat IoT. Dalam proses perangkat IoT melakukan *update* lalu *restart*, inilah terjadi perbedaan waktu *delay*. karena ketika proses *update firmware* ini terjadi *update firmware* gagal dan *update firmware* berhasil. Jika *update firmware* berhasil proses akan lebih cepat dari pada ketika proses *update firmware* gagal, dengan hasil pengujian *delay* sebagai berikut.

Tabel 3. *Delay* pengiriman *firmware*, *update firmware*, dan *restart* perangkat IoT

QoS	Delay(s) Sending *Firmware* with *Process Update* & *Restart*
QoS 0	29.684037
QoS 1	39.807739
QoS 2	37.997534

Tabel 3 adalah *delay* pada saat *publish* yang dilakukan aplikasi berbasis *website* dan diterima oleh perangkat IoT, ditambah dengan proses *update firmware* dan *restart* yang dilakukan perangkat IoT. *Delay* ketika menggunakan QoS 0 dengan proses *update firmware* dan *restart* didapatkan *delay* 29.684037/s, jauh berbeda dengan nilai *delay* QoS 0 *delay* pengiriman *firmware*, dengan nilai *delay* 0.105521/s, perbedaan

ini terjadi karena nilai *delay* QoS 0 ditambah dengan proses *update firmware* dan *restart* perangkat IoT lalu mengirimkan paket *publish* berisi *version update firmware*.

Pada *delay* QoS 1 dan QoS 2 memiliki nilai *delay* yang meningkat dari pada *delay* pengiriman *firmware*. Hal ini terjadi karena ada proses *update firmware* dan *restart* perangkat IoT pada saat menerima *file firmware*, proses *update firmware* dan *restart* perangkat IoT dapat terjadi berhasil dan gagal *update firmware* ke perangkat IoT, jika proses *update firmware* berhasil maka *delay* akan menjadi lebih cepat, daripada jika gagal akan menjadi lebih lama. Dari *delay* proses pengiriman *file firmware* dikurang dengan *delay* proses *update firmware* dan *restart* perangkat IoT didapatkan hasil *delay* tertinggi pada masing-masing QoS 0, QoS 1, dan QoS 2 sebagai berikut.

Tabel 4. Hasil delay pengiriman *firmware* dikurangi *delay* proses *update* dan *restart*.

QoS	*Delay Sending Firmware - Delay with Process Update and Restart*
QoS 0	29.578506
QoS 1	19.702856
QoS 2	17.863652

Tabel 4 adalah hasil pengurangan *delay* pada Tabel 3, untuk melihat *delay* tertinggi pada saat aplikasi berbasis *website publish firmware* lalu diterima oleh perangkat IoT, dan melakukan *update firmware* kemudian melakukan *restart* perangkat IoT. Pada QoS 0 memiliki nilai *delay* tertinggi 29.578506/s karena mengikuti tingkat kegagalan pada saat proses *update firmware* yang hanya terkirim 50%, berikutnya QoS 1 *delay* tertinggi kedua 19.702856/s dengan tingkat kegagalan proses *update* 30% dan pada QoS 2 memiliki tingkat *delay* 17.663625/s dengan tingkat kegagalan proses u*pdate* 20%. Dari proses *update firmware* yang berhasil dan gagal pada saat proses *update firmware* dilanjutkan *restart* perangkat IoT dapat disimpulkan bahwa pada saat *update firmware* perangkat IoT, penggunaan QoS 2 lebih disarankan dari pada QoS 1 dan QoS 0, karena tingkat kegagalan pengiriman *update firmware* menggunakan QoS 2 = 20% atau 2 kali kegagalan dari 10 kali percobaan pengiriman, namun penggunaan QoS 2, juga berpengaruh ke pada penggunaan sumber daya CPU dan *memory* paling tinggi dari pada QoS 0, dan QoS 1.

4. Kesimpulan

Dari hasil pengujian pengiriman *file firmware* untuk *over the air update firmware* perangkat IoT didapatkan hasil direkomendasi menggunakan QoS 2 karena tingkat keberhasilan pengiriman *file firmware* dalam 10 kali percobaan adalah 80 % dari pada QoS 1 = 70 % dan QoS 0 = 50 %. Penggunaan QoS 2 pada saat mengirim *file firmware* mengakibatkan penggunaan sumber daya CPU dan *memory* juga tinggi dari pada menggunakan QoS 1 dan QoS 0. Tingkat kegagalan QoS 0 = 50% , QoS 1 = 70 %, dan QoS 2 = 80% berpengaruh terhadap *delay* pada saat mengirim *file firmware*. Dengan *delay* tertinggi QoS 0 = 29.578506/s, diikuti QoS 1 = 19.702856/s ,dan delay terendah QoS 2 = 17.663625/s.

Daftar Pustaka

[1] D. Lund and M. Morales, “Worldwide and Regional Internet of Things (IoT) 2014 – 2020 Forecast : A Virtuous Circle of Proven Value and Demand,” *IDC Anal. Futur.*, no. May, p. 29, 2014.
[2] D. Evans, “The Internet of Things How the Next Evolution of the Internet Is Changing Everything,” 2011.
[3] S. Reißmann and C. Pape, “An Over the Air Update Mechanism for ESP8266 Microcontrollers,” *ICSNC 2017 Twelfth Int. Conf. Syst. Networks Commun. An*, no. October, pp. 11–17, 2017.
[4] Lee Jeffrey, “Over-The-Air Firmware: The Critical Driver of IoT Success - DZone IoT,” *https://dzone.com*, 2017.
[5] ESP8266, “OTA Update · ESP8266 Arduino Core.”
[6] A. S. A. Quadri and B. . O. Sidek, “An Introduction to Over-the-Air Programming in Wireless Sensor Networks,” *Int. J. Comput. Sci. Netw. Solut.*, vol. 2, pp. 33–49, 2014.
[7] M. K. H. R A Atmoko*, R Riantini, “IoT real time data acquisition using MQTT protocol,” *Int. Conf. Phys. Instrum. Adv. Mater.*, vol. 012003, 2016.
[8] N. A. Stanford Clark Andy, “MQTT,” *IBM*, 1999.
[9] Zhang Lucy, “Building Facebook Messenger,” 2011.