Message Queuing Telemetry Transport dalam Internet of Things Menggunakan ESP-32

Darian Rizaludin¹, Yulius Satmoko Raharjo², Aryo Nugroho³, Moh Noor Al-Azam^{4*}

¹ Fakultas Ilmu Komputer, Program Studi Sistem Komputer, Universitas Narotama, Surabaya, Indonesia

^{2,3} Fakultas Ilmu Komputer, Program Studi Sistem Informasi, Universitas Narotama, Surabaya, Indonesia

⁴ Fakultas Ilmu Komputer, Program Studi Teknik Informatika, Universitas Narotama, Surabaya, Indonesia

Email: ¹darian@rad.net.id, ²yulius.satmoko@narotama.ac.id, ³aryo.nugroho@narotama.ac.id, ^{4*}noor.azam@narotama.ac.id

Abstrak

Message Queuing Telemetry Transport (MQTT) adalah protokol konektivitas antar mesin atau yang sekarang lebih dikenal dengan Internet of Things (IoT). Protokol ini mengenal dua fungsi dasar komunikasi M2M, yaitu publish dan subscribe (pub/sub). Protokol MQTT didisain sebagai protokol pengiriman pesan dengan sangat sederhana dan sangat ringan, dirancang untuk perangkat yang terbatas dan dengan kapasitas bandwidth yang rendah, berlatensi tinggi atau pada sebuah jaringan tidak dapat diandalkan. Prinsip-prinsip desainnya adalah untuk meminimalkan kebutuhan bandwidth dan kebutuhan sumber daya perangkat, dan tetap berusaha untuk memastikan kehandalan dan tingkat jaminan pengiriman.

Dalam tulisan ini diujicoba kehandalan performansi VerneMQ -salah satu MQTT broker, dengan beberapa tingkat stressing dengan menggunakan ESP-32 sebagai publisher dan notebook dengan aplikasi python sebagai subscriber.

Kata Kunci: IoT, M2M, MQTT, Message Queuing Telemetry Transport, ESP-32

Abstract

Message Queuing Telemetry Transport (MQTT) is a connectivity protocol between machines or now better known as the Internet of Things (IoT). This protocol recognizes two basic functions of M2M communication, namely publish and subscribe (pub/sub). The MQTT protocol is designed as a very simple and very lightweight message delivery protocol, designed for devices that are limited and with low bandwidth capacity, high latency or on an unreliable network. The design principles are to minimize bandwidth requirements and device resource requirements, and keep trying to ensure reliability and guaranteed delivery rates. In this paper, VerneMQ performance reliability is tested - one of the MQTT brokers, with several stressing levels using ESP-32 as a publisher and notebook with the python application as a subscriber.

Keywords: IoT, M2M, MQTT, Message Queuing Telemetry Transport, ESP-32

1. PENDAHULUAN

Kebutuhan komunikasi antar mesin, atau yang lebih dikenal dengan istilah *Machine-to-Machine* (M2M) masih akan terus berkembang pesat seiring dengan perkembangan Teknologi Informasi dan Komunikasi (TIK) dan kebutuhan di era *Internet of Things* (IoT) dan Industri 4.0 dewasa ini.

Dengan semakin murahnya biaya komputasi secara global membuat ketersediaan berbagai macam sensor dan *actuator* yang juga semakin murah dan semakin kecil untuk dapat dimasukkan dalam berbagai perangkat bergerak atau *mobile computing*. Kemajuan teknologi yang luar biasa dalam komputasi awan (*cloud computing*) juga membuat media penyimpanan dan analisa data yang dihasilkan oleh sensor yang tertanam di berbagai peralatan menjadi semakin mudah dan murah. Pada akhirnya, dengan dipicu oleh konektivitas yang berada hampir di semua lokasi dan ketersediaan alamat IP yang hampir tidak terbatas dengan IPv6 membuat kenaikan perangkat yang terkoneksi dengan Internet akan menyentuh angka 25 milyar atau lebih pada tahun 2020 yang akan datang (gambar 1) [1].

Gambar 1. Pertumbuhan Connected Devices [1]

Dalam dasar ilmu komunikasi digital, komunikasi antar perangkat ini sudah pasti akan menggunakan model standar *Open System Interconnection* (OSI) Layer [2], yang OSI layer ini juga diadopsi oleh protokol Internet yang saat ini secara umum digunakan (gambar 2).

Gambar 1. OSI Layer

Salah satu kesulitan dalam penggunaan koneksi *Internet Protocol* (IP) *base* dalam sebuah komunikasi IoT adalah saat dibutuhkannya pengiriman perintah dari jarak jauh untuk *actuator* agar melakukan suatu tugas. Hal ini karena konsep koneksi *client/server* yang digunakan dalam IP harus dimulai dari sisi *client*, sementara yang menerima perintah atau sisi server harus selalu dalam posisi *listen* pada nomor *port* tertentu agar dapat menerima permintaan koneksi dari *client*.

Dengan konsep *client/server* ini maka pada perangkat *actuator* harus dilakukan pemrograman dalam mode server, dengan alasan agar dapat diperintah setiap saat tersebut. Sementara itu pemrograman dalam mode server akan lebih banyak menghabiskan sumber daya komputasi sehingga tidak lagi sesuai dengan kondisi dan tujuan IoT itu sendiri.

Selain itu, dengan konsep *client/server* maka pada perangkat *actuator* harus bisa diakses secara langsung dari sebuah titik perangkat yang memberikan perintah tersebut. Hal ini juga merupakan sebuah permasalahan tersendiri untuk pengembangan pada saat ini, dimana masih banyak jaringan yang menggunakan IP *private* dengan *Network Address Translation* (NAT) pada jaringan-jaringan selular dan jaringan publik lainnya. Dan seperti diketahui, dengan menggunakan NAT maka IP *address* yang berada di dalam sebuah NAT tidak akan bisa dijangkau secara langsung dari IP *address* yang berada di luarnya.

Untuk mengatasi hal-hal di atas, pada akhir tahun 1990 IBM mengembangkan sebuah protokol komunikasi yang diberi nama *Message Queuing Telemetry Transport* (MQTT) [1]. Protokol yang dibangun di atas *Transmission Control Protocol* (TCP) ini, secara praktis telah menjadi standar komunikasi untuk M2M, IoT, dan Wireless Sensor Network (WSN) [3].

2. TEORITIS

2.1 Message Queuing Telemetry Transport

MQTT adalah protokol pengiriman pesan yang sangat sederhana dan ringan, serta dirancang untuk digunakan dalam perangkat dengan sumber daya terbatas dengan kapasitas bandwidth rendah, latensi tinggi atau pada jaringan yang tidak dapat diandalkan.

Prinsip-prinsip desain MQTT adalah untuk meminimalkan bandwidth jaringan dan kebutuhan sumber daya perangkat namun tetap berusaha untuk memastikan keandalan dan beberapa tingkat jaminan pengiriman data. Prinsip-prinsip ini juga berubah untuk membuat protokol ideal dari kebutuhan untuk M2M atau untuk IoT yang muncul dari perangkat yang terhubung, dan untuk aplikasi perangkat bergerak yang memiliki bandwidth dan daya baterai sangat terbatas.

Gambar 2. pub/sub MQTT

MQTT mengenal 2 fungsi dasar: *publish* dan *subscribe* (Gambar 3). *Publish* adalah fungsi yang digunakan oleh perangkat mengirimkan data-data yang dimilikinya atau perintah-perintah yang ingin dikirimkan ke perangkat

lainnya. Sedangkan fungsi subscribe adalah fungsi yang digunakan untuk membaca data atau perintah yang dikirimkan oleh perangkat lain yang harus diketahuinya.

Semua data yang di-publish oleh sebuah perangkat dalam MQTT akan dikirimkan ke sebuah server MQTT broker. Sesuai dengan namanya, sebuah server MQTT broker hanya menerima data yang didapatkan dari sebuah perangkat (publisher) dan kemudian meneruskan data itu ke perangkat-perangkat lainnya (subscriber). Server MQTT broker secara standar tidak menyimpan data-data yang dilaporkan oleh perangkat publisher. Server MQTT broker hanya menerima data yang dilaporkan dan kemudian meneruskannya ke perangkat publisher yang dikenalnya.

Satu buah perangkat dapat menggunakan kedua fungsi publish dan subscribe di atas secara bersamaan. Misalkan pada sebuah perangkat bergerak yang perlu menerima laporan dari sebuah sensor dan juga perlu mengirimkan perintah pada sebuah actuator. Atau sebuah perangkat embedded system yang berfungsi sebagai sensor dan actuator -sehingga dia perlu mengirimkan data sensor dan menerima perintah untuk actuator.

Untuk membedakan data-data yang diterima sebuah dari publisher guna diteruskan ke subscriber yang mana, di dalam sistem MQTT dikenal adanya topic.

Topic MQTT disusun dalam sebuah susunan bertingkat yang mirip dengan folder atau direktori dan file dalam sistem file yang kita kenal, dengan menggunakan garis miring (/) sebagai pembatasnya. Dengan menggunakan sistem seperti ini, maka dapat membuat struktur penamaan yang ramah pengguna dan deskriptif sesuai pilihan.

Penamaan topic MQTT mengikuti aturan:

1. Case sensitive atau dibedakan antara huruf kapital dan non-kapital.
2. Menggunakan standar string UTF-8.
3. Minimal memiliki satu karakter.

Selain topic, MQTT juga memiliki sebuah quality of service (QoS) [3][4].

QoS ini adalah sebuah kesepakatan antara publisher dan subscriber tentang jaminan pengiriman pesan. Ada tiga level QoS: 0 – minimal sekali

1 – setidaknya sekali

2 – hanya sekali

Saat sebuah perangkat publisher (mis. Perangkat IoT jarak jauh) mengirimkan data ke MQTT broker, perangkat tersebut menentukan tingkat QoS untuk pesan itu. Ketika MQTT broker mengirim pesan ke perangkat subscriber, QoS yang ditetapkan oleh publisher untuk pesan itu digunakan lagi.

Jadi secara efektif publisher yang menentukan QoS data sampai ke penerima akhir [4] [5]:

- a) QoS Level 0 – minimal satu kali

QoS ini adalah QoS yang paling sederhana, overhead pengiriman pesan yang terendah. Perangkat publisher hanya mengirimkan data atau pesan satu kali, dan tidak ada pengakuan oleh MQTT broker.

- b) QoS Level 1 – setidaknya satu kali

QoS ini akan menjamin bahwa pesan akan berhasil ditransfer ke MQTT broker. MQTT broker akan mengirim acknowledgement keberhasilan ke publisher, tetapi jika acknowledgement hilang, maka publisher tidak akan menyadari pesannya telah diterima oleh MQTT broker, jadi publisher akan mengirim pesan lagi. Publisher akan terus mengirim ulang sampai mendapat acknowledgement dari MQTT broker.

Ini berarti bahwa pengiriman data atau pesan dijamin, meskipun data atau pesan tersebut dapat diterima oleh MQTT broker lebih dari satu kali.

- c) QoS Level 2 - tepat satu kali

QoS ini adalah tingkat QoS tertinggi, sehingga memerlukan adanya empat tahap antara publisher dan subscriber, untuk mengonfirmasi bahwa data atau pesan telah dikirim dan bahwa acknowledgement telah diterima. Ketika handshake sebagai penanda data atau pesan telah dikirimkan dan diterima selesai, maka dapat diyakinkan bahwa data atau pesan itu dikirim tepat satu kali saja.

Dari ketiga QoS yang dimiliki oleh sistem MQTT di atas, mana yang akan paling tepat digunakan untuk aplikasi M2M atau IoT?

Ada beberapa pendapat tentang hal ini, dan tidak satu aturan yang pasti yang menyatakan bahwa satu QoS lebih baik daripada yang lain. Pemilihan QoS hanya berdasarkan dari pengalaman atau best practice yang ada kemungkinan berbeda antara satu kebutuhan dengan kebutuhan yang lain.

QoS 0 adalah QoS dengan biaya terendah dalam hal volume transfer data. QoS ini sesuai digunakan ketika koneksi yang tersedia andal antara perangkat dan MQTT broker. Harus dipertimbangkan apakah sistem atau aplikasi IoT dapat mentoleransi hilangnya data atau pesan. Sebagai contoh, seandainya perangkat memonitor sesuatu dan mengirimkan pembacaan kumulatif, maka dampak dari pesan yang hilang hanya akan menjadi keterlambatan dalam pembacaan yang mencapai server.

QoS 1 adalah pilihan QoS yang baik jika diperlukan kepastian setiap data atau pesan berhasil dikirimkan, namun sistem atau aplikasi IoT harus dapat mentolerir penerimaan data atau pesan lebih dari satu kali. Jika publisher pengirim pembacaan data kumulatif dari suatu sensor, maka QoS 1 mungkin cocok, karena setiap data yang diterima lebih dari satu kali oleh subscriber memiliki efek yang kecil. Tetapi jika bacaan yang dikirim

publisher adalah beberapa perubahan sejak pengiriman terakhir, maka data duplikat dapat membuat data yang menyestakan dalam sistem kita.

Salah satu cara untuk mengatasi kesalahan ini adalah memastikan perangkat mengirim setiap pesan dengan timestamp yang unik setiap waktu dan di sisi subscriber harus dikenali data duplikasi tersebut berdasarkan timestamp dan mengabaikannya.

QoS 2 menjamin pengiriman tepat hanya satu kali, tetapi memiliki biaya yang relatif tinggi dalam hal transfer data karena ada mekanisme handshake yang harus dilakukan.

Kebanyakan aplikasi akan bekerja tanpa masalah dengan QoS 2, tetapi harus dipertimbangkan apakah QoS yang lebih murah masih dapat menangani atau memang harus QoS ini. Seringkali melalui perencanaan yang cermat dalam penggunaan data yang akan dikirimkan -misalkan menggunakan timestamp atau pembacaan data secara kumulatif - adalah cara mudah untuk menghindari overhead dalam QoS 2.

3. ANALISA DAN PEMBAHASAN

3.1 Topologi dan Perangkat Keras

Dalam percobaan ini, dilakukan simulasi dengan topologi jaringan seperti yang dilakukan pada gambar 4. Pada sisi IoT atau embedded system, digunakan ESP-32 keluaran dari Espressif System yang sudah dilengkapi dengan WiFi. Pada percobaan nantinya akan dikoneksikan ke Internet melalui sebuah Access Point (AP) dengan menggunakan kartu 3G [6].

IoT ini akan melakukan publish ke MQTT broker milik Universitas Narotama yang terletak di Data Center RADNET yang terkoneksi langsung dengan backbone Internet dengan bandwidth 100 Mbps -untuk menjamin tidak ada masalah komunikasi dari Internet ke MQTT broker.

Terakhir, sebuah notebook digunakan sebagai perangkat untuk subscribe pada MQTT broker dengan topic yang sama dengan publisher ESP-32. Notebook ini terkoneksi dengan jaringan Fiber To The Home (FTTH) yang memiliki kapasitas koneksi 30 Mbps. Laporan yang dihasilkan dari subscriber ini akan digunakan sebagai dasar analisa kehandalan topologi sistem MQTT di atas.

3.2 Perangkat Lunak – MQTT broker

MQTT broker server menggunakan perangkat keras dengan processor Xeon x3100 M4 3.1GHz/1600MHz/8MB dan diperkuat memori RAM sebesar 4 GB. Menggunakan sistem operasi Ubuntu Linux 18.04.2 LTS dan dengan menggunakan VerneMQ versi 1.6.1 sebagai MQTT broker.

MQTT broker ini sama sekali tidak melakukan penyimpanan data pada server, oleh karenanya pada percobaan ini diperlukan juga sebuah subscriber untuk menghitung dan menganalisa seberapa besar kesalahan terjadi dalam pengiriman data atau pesan publisher pada MQTT broker [7].

3.3 Perangkat Lunak – ESP-32

ESP-32 adalah perangkat sistem tertanam buatan Espressif System. Pada sistem ini tertanam dual-core 32-bit LX6 microprocessor, dengan kecepatan 600MHz. Sistem juga dilengkapi dengan 448 KB ROM dan 520 KB SRAM [8].

Gambar 3. Topologi Penelitian

Jauh sebelum ada ESP32, dunia mengenal adanya Arduino. Salah satu daya tarik Arduino adalah kompleksitasnya yang relatif rendah yang memberikan hampir setiap orang kemampuan untuk membangun sesuatu dengan cepat dan mudah. Integrated Development Environment (IDE) untuk Arduino juga mudah dan selalu gratis untuk diunduh dari Internet [9]. Dalam percobaan ini, digunakan IDE Arduino untuk membuat publisher di dalam ESP-32.

Di dalam ESP-32 ini dibuatkan program dengan flow chart seperti pada gambar 5.

Program dimulai pada fungsi setup dengan tugas utama melakukan koneksi pada WiFi. Untuk pekerjaan ini diperlukan modul WiFi.h [6] [10]. Setelah ESP-32 berhasil terkoneksi pada AP, pekerjaan dilanjutkan dengan membuat nilai awal variabel counter -yang akan digunakan sebagai data yang akan dikirimkan melalui publish MQTT, dan delay -yang akan digunakan sebagai nilai delay pada ESP-32.

Selanjutnya program menaikkan nilai variabel counter, menyimpan data-data yang diperlukan dalam JSON, membangun koneksi dengan MQTT broker, dan melakukan publish data JSON. Langkah-langkah ini diulangi terus sampai dengan variable counter mencapai nilai 100.

Saat nilai variabel counter mencapai nilai 100, maka variabel counter direset kembali menjadi 0, dan variable delay akan dikurangi 10. Kedua langkah di atas ini diulangi terus hingga variabel delay mencapai nilai 0 dan berhenti.

3.4 Perangkat Lunak – Notebook

Sebagai *subscriber* MQTT dalam percobaan ini digunakan sebuah notebook i5/1.6 GHz dengan memori RAM sebesar 4 GB. Sebuah aplikasi dibuat dengan menggunakan bahasa pemrograman python dan mengikuti *flowchart* gambar 6.

Untuk koneksi dengan MQTT *broker*, aplikasi menggunakan library dari *Paho MQTT Client* dari *Eclipse*. Proyek Eclipse Paho menyediakan implementasi klien open-source protokol MQTT dan MQTT-SN yang ditujukan untuk aplikasi baru, yang sudah ada, dan yang baru muncul untuk Internet of Things (IoT) [11].

Pertama yang dilakukan oleh aplikasi adalah koneksi ke MQTT broker dan menunggu adanya data yang dikirimkan ke topic yang sudah menjadi kesepakatan sebelumnya.

Saat data diterima pada topic tersebut, maka yang dilakukan pertama adalah mencatat timestamp kedatangan data tersebut -karena secara standar ESP-32 tidak memiliki Real Time Clock (RTC) sehingga data yang dikirimkan ke MQTT broker tidak dapat diberikan timestamp di sisi ESP-32. Oleh karenanya, pemberian timestamp dilakukan pada sisi subscriber.

Timestamp ini digunakan untuk menghitung berapa rata-rata data yang diterima oleh subscriber dalam 1 menit. Setelah timestamp didapatkan, data disimpan ke dalam sebuah file untuk analisa lebih lanjut.

Penerimaan dan penyimpanan data ini dilakukan terus selama data diterima dan sampai data terakhir yang dipublish oleh ESP-32.

Gambar 4. flowchart program ESP-32

Gambar 5. flowchart program subscriber

4. IMPLEMENTASI

Hasil dari percobaan di atas menghasilkan sebuah file yang berisi urutan nomor dari 0 sampai dengan 99 (100 buah data) dan dikirimkan dengan jarak pengiriman (delay) dari 1000 mS sampai dengan tanpa delay. Contoh data yang kita dapatkan dapat dilihat pada gambar 7.

```

STARTING NOW!!!
{"data":{"time":"Sat Mar 16 15:44:14 2019","mac":"24:8A:C4:AE:80:90","count":"0","delay":"1000"}}
{"data":{"time":"Sat Mar 16 15:44:15 2019","mac":"24:8A:C4:AE:80:90","count":"1","delay":"1000"}}
{"data":{"time":"Sat Mar 16 15:44:16 2019","mac":"24:8A:C4:AE:80:90","count":"2","delay":"1000"}}
{"data":{"time":"Sat Mar 16 15:44:17 2019","mac":"24:8A:C4:AE:80:90","count":"3","delay":"1000"}}
{"data":{"time":"Sat Mar 16 15:44:18 2019","mac":"24:8A:C4:AE:80:90","count":"4","delay":"1000"}}
{"data":{"time":"Sat Mar 16 15:44:19 2019","mac":"24:8A:C4:AE:80:90","count":"5","delay":"1000"}}
{"data":{"time":"Sat Mar 16 15:44:20 2019","mac":"24:8A:C4:AE:80:90","count":"6","delay":"1000"}}
{"data":{"time":"Sat Mar 16 15:44:21 2019","mac":"24:8A:C4:AE:80:90","count":"7","delay":"1000"}}
{"data":{"time":"Sat Mar 16 15:44:22 2019","mac":"24:8A:C4:AE:80:90","count":"8","delay":"1000"}}
{"data":{"time":"Sat Mar 16 15:44:23 2019","mac":"24:8A:C4:AE:80:90","count":"9","delay":"1000"}}
{"data":{"time":"Sat Mar 16 15:44:24 2019","mac":"24:8A:C4:AE:80:90","count":"10","delay":"1000"}}
{"data":{"time":"Sat Mar 16 15:44:25 2019","mac":"24:8A:C4:AE:80:90","count":"11","delay":"1000"}}
{"data":{"time":"Sat Mar 16 15:44:26 2019","mac":"24:8A:C4:AE:80:90","count":"12","delay":"1000"}}
{"data":{"time":"Sat Mar 16 15:44:27 2019","mac":"24:8A:C4:AE:80:90","count":"13","delay":"1000"}}
{"data":{"time":"Sat Mar 16 15:44:28 2019","mac":"24:8A:C4:AE:80:90","count":"14","delay":"1000"}}
{"data":{"time":"Sat Mar 16 15:44:29 2019","mac":"24:8A:C4:AE:80:90","count":"15","delay":"1000"}}
{"data":{"time":"Sat Mar 16 15:44:30 2019","mac":"24:8A:C4:AE:80:90","count":"16","delay":"1000"}}
{"data":{"time":"Sat Mar 16 15:44:31 2019","mac":"24:8A:C4:AE:80:90","count":"17","delay":"1000"}}
{"data":{"time":"Sat Mar 16 15:44:32 2019","mac":"24:8A:C4:AE:80:90","count":"18","delay":"1000"}}
{"data":{"time":"Sat Mar 16 15:44:33 2019","mac":"24:8A:C4:AE:80:90","count":"19","delay":"1000"}}
{"data":{"time":"Sat Mar 16 15:44:34 2019","mac":"24:8A:C4:AE:80:90","count":"20","delay":"1000"}}
{"data":{"time":"Sat Mar 16 15:44:35 2019","mac":"24:8A:C4:AE:80:90","count":"21","delay":"1000"}}

```

Gambar 6. Data JSON *published* dari ESP-32 yang diterima *subscriber*

Data flat file yang dihasilkan oleh aplikasi subscriber seperti pada gambar 7, selanjutnya kita lakukan parsing sesuai data JSON yang ada. Hasil dari parsing dilanjutkan dengan memeriksa apakah ada variabel count yang terlewat atau tidak. Variabel ini kita gunakan sebagai penanda bahwa ada 100 buah data yang dipublish dari ESP-32 dan seharusnya diterima oleh aplikasi subscriber di notebook. Jika variabel count urut diterima seluruhnya -dari 0 sampai dengan 99, maka artinya seluruh pengiriman sukses dilaksanakan. Pemeriksaan variabel count ini kemudian dibuatkan persentase sukses. Jika 100% maka artinya seluruh data yang dikirimkan oleh ESP-32 melalui publish ke MQTT broker dan diterima oleh notebook melalui subscribe, berhasil diterima seluruhnya. Jika ada data yang tidak diterima, maka persentase sukses akan menyesuaikan. Analisa selanjutnya adalah menghitung berapa waktu yang diperlukan oleh subscriber menerima 100 data dengan delay yang sama tersebut. Waktu yang diperlukan untuk setiap delay dapat dilihat pada gambar 8.

Gambar 7. Data JSON *published* dari ESP-32

Pada gambar tersebut terlihat semakin kecil delay maka semakin cepat waktu yang diperlukan untuk publish. Sementara pada gambar 9 dapat dilihat korelasi waktu yang dibutuhkan untuk mengirimkan per-100 data atau pesan oleh *publisher*, dengan persentasi penerimaan data atau pesan oleh *subscriber*.

Gambar 8. Hasil Akhir Stressing Test

Penghitungan ini dilakukan dengan cara mengambil nilai timestamp awal data diterima, yaitu saat data variabel count bernilai 0, dan nilai timestamp akhir data diterima, saat data variabel count bernilai 99. Dari dua data

timestamp ini, kemudian dihitung selisih waktunya. Selisih waktu tersebut adalah waktu yang diperlukan untuk mengirimkan 100 data

Dari data-data dan perhitungan yang dilakukan di atas, maka kita memiliki data transmisi pub/sub MQTT dengan beberapa nilai delay, yang dapat dilihat pada tabel 1.

Tabel 1. Hasil Data Percobaan

Delay	Success Rate	Time	Data per-S	Delay	Success Rate	Time	Data per-S
1000	100%	00:01:39	1,01	500	100%	00:00:50	2,00
990	100%	00:01:38	1,02	490	100%	00:00:49	2,04
980	100%	00:01:37	1,03	480	100%	00:00:48	2,08
970	100%	00:01:36	1,04	470	100%	00:00:47	2,13
960	100%	00:01:36	1,04	460	100%	00:00:46	2,17
950	100%	00:01:34	1,06	450	100%	00:00:45	2,22
940	100%	00:01:33	1,08	440	100%	00:00:43	2,33
930	100%	00:01:32	1,09	430	100%	00:00:42	2,38
920	100%	00:01:31	1,10	420	100%	00:00:42	2,38
910	100%	00:01:30	1,11	410	100%	00:00:41	2,44
900	100%	00:01:29	1,12	400	100%	00:00:40	2,50
890	100%	00:01:28	1,14	390	100%	00:00:40	2,50
880	100%	00:01:27	1,15	380	100%	00:00:37	2,70
870	100%	00:01:27	1,15	370	100%	00:00:37	2,70
860	100%	00:01:26	1,16	360	100%	00:00:36	2,78
850	100%	00:01:24	1,19	350	100%	00:00:35	2,86
840	100%	00:01:23	1,20	340	100%	00:00:35	2,86
830	100%	00:01:22	1,22	330	100%	00:00:31	3,23
820	100%	00:01:21	1,23	320	100%	00:00:32	3,13
810	100%	00:01:20	1,25	310	100%	00:00:31	3,23
800	100%	00:01:19	1,27	300	100%	00:00:30	3,33
790	100%	00:01:18	1,28	290	100%	00:00:29	3,45
780	100%	00:01:18	1,28	280	100%	00:00:28	3,57
770	100%	00:01:17	1,30	270	100%	00:00:27	3,70
760	100%	00:01:15	1,33	260	100%	00:00:26	3,85
750	100%	00:01:14	1,35	250	100%	00:00:25	4,00
740	100%	00:01:13	1,37	240	100%	00:00:23	4,35
730	100%	00:01:12	1,39	230	100%	00:00:23	4,35
720	100%	00:01:11	1,41	220	100%	00:00:22	4,55
710	100%	00:01:10	1,43	210	100%	00:00:21	4,76
700	100%	00:01:09	1,45	200	100%	00:00:20	5,00
690	100%	00:01:06	1,52	190	100%	00:00:19	5,26
680	100%	00:01:08	1,47	180	100%	00:00:18	5,56
670	100%	00:01:07	1,49	170	100%	00:00:17	5,88
660	100%	00:01:05	1,54	160	100%	00:00:16	6,25
650	100%	00:01:05	1,54	150	100%	00:00:16	6,25
640	100%	00:01:03	1,59	140	100%	00:00:14	7,14
630	100%	00:01:02	1,61	130	100%	00:00:13	7,69
620	100%	00:01:01	1,64	120	100%	00:00:12	8,33
610	100%	00:01:00	1,67	110	100%	00:00:11	9,09
600	100%	00:01:00	1,67	100	100%	00:00:10	10,00
590	100%	00:00:59	1,69	90	100%	00:00:10	10,00
580	100%	00:00:58	1,72	80	100%	00:00:08	12,50
570	100%	00:00:59	1,69	70	100%	00:00:06	16,67
560	100%	00:00:54	1,85	60	100%	00:00:06	16,67
550	100%	00:00:55	1,82	50	100%	00:00:05	20,00
540	100%	00:00:53	1,89	40	100%	00:00:04	25,00
530	100%	00:00:52	1,92	30	100%	00:00:03	33,33
520	100%	00:00:51	1,96	20	100%	00:00:02	50,00
510	100%	00:00:51	1,96	10	100%	00:00:02	50,00
500	100%	00:00:50	2,00	0	100%	00:00:01	100,00

Tabel ini kemudian dilengkapi dengan perhitungan data per-second untuk masing-masing nilai delay pada saat dilakukan percobaan pub/sub. Nilai ini diambilkan dari 100 data dibagi dengan nilai data waktu yang dibutuhkan.

Penghitungan throughput data per-second dilakukan dengan mengubah nilai waktu yang diperlukan menjadi nilai numerik seperti pada rumus (1).

$$s = 24 \times 60 \times 60 \times Time \quad (1)$$

Nilai dalam variabel *Time* ini disimpan dalam bentuk numerik dengan nilai 1 adalah 24 jam atau 86.400 detik. Oleh karenanya kita lakukan pengubahan nilai menjadi detik seperti pada rumus (1) Nilai detik (s) ini kemudian digunakan sebagai acuan dengan membagi data yang diterima dengan s (1)

$$\text{Throughput} = \frac{(100 \times \text{Success Rate})}{S} \quad (2)$$

Nilai *Throughput* dapat dilihat dalam tabel 1 pada kolom data per-S.

5. KESIMPULAN

Dari percobaan yang dilakukan, VerneMQ sebagai MQTT broker dengan ESP-32 menggunakan jaringan 3G sebagai publisher dapat menangani lebih dari 100 data atau pesan per-detik untuk kebutuhan transmisi data di IoT. Kemampuan ini diperkirakan masih jauh di atas, karena dari notebook sebagai subscriber belum menemukan adanya kehilangan data -meski publish yang dilakukan hanya menggunakan QoS 0, yang merupakan QoS paling rendah dari sistem MQTT. Oleh karenanya, penggunaan MQTT sebagai salah satu EcoSystem IoT untuk komunikasi M2M pada daerah dengan sinyal 3G di Indonesia layak untuk digunakan.

REFERENCES

- [1] IBM Institute for Business Value, "Device democracy: Saving the Future of The Internet of Things." IBM Corporation, Jul-2015.
- [2] M. N. Al-Azam, M. M. Achlaq, A. Nugroho, A. Gabriel Sooi, A. Winaya, and Maftuchah, "Broadcasting the Status of Plant Growth Chamber using Bluetooth Low Energy," MATEC Web Conf., vol. 164, p. 01029, 2018.
- [3] R. Giambona, A. E. C. Redondi, and M. Cesana, "MQTT+: Enhanced Syntax and Broker Functionalities for Data Filtering, Processing and Aggregation," in Proceedings of the 14th ACM International Symposium on QoS and Security for Wireless and Mobile Networks - Q2SWinet '18, Montreal, QC, Canada, 2018, pp. 77–84.
- [4] D. Chattopadhyay, A. Samantaray, and A. Datta, "Device microagent for IoT home gateway: a lightweight plug-n-play architecture," ACM SIGBED Rev., vol. 15, no. 2, pp. 16–23, Jun. 2018.
- [5] P. Colombo and E. Ferrari, "Access Control Enforcement within MQTT-based Internet of Things Ecosystems," in Proceedings of the 23rd ACM on Symposium on Access Control Models and Technologies - SACMAT '18, Indianapolis, Indiana, USA, 2018, pp. 223–234.
- [6] Ricardo Haryunarendra, Darian Rizaluddin, and Moh Noor Al-Azam, "PERFORMA JARINGAN FREE WIRELESS DI TAMAN KOTA SURABAYA," J. LINK, vol. 26, pp. 5–25, Sep. 2017.
- [7] R. Giambona, A. E. C. Redondi, and M. Cesana, "Demonstrating MQTT+: An Advanced Broker for Data Filtering, Processing and Aggregation," in Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems - MSWIM '18, Montreal, QC, Canada, 2018, pp. 357–358.
- [8] Mochamad Mizanul Achlaq, Moh Noor Al-Azam, Maftuchah Maftuchah, and Aris Winaya, "Nawala Mangsa 2.0 – Weather Station with BLE Broadcaster," Univ. Islam Sultan Agung, vol. 6, no. 2, Jun. 2018.
- [9] Neil Kolban, Kolban's Book on ESP 32. Victoria, British Columbia, Canada: Lean Publishing., 2018.
- [10] D. Rachman, M. N. Al Azam, and B. Anindito, "Sistem Pemantau & Pengendalian Rumah Cerdas Menggunakan Infrastruktur Internet Messaging," J. Ilm. Link, vol. 26, no. 1, p. 34, 2017.
- [11] G. C. Hillar, Internet of Things with Python: Interact with the world and rapidly prototype IoT applications using Python. Birmingham Mumbai: Packt Publishing Limited, 2016.