PENERAPAN MOSQUITTO SEBAGAI SERVER MQTT LOKAL PADA KOMUNIKASI ANTARA *DATABASE* MONGODB DAN MESIN ABSENSI *MULTI-NODE*

Khiarul Arham, Agus Wagyana
Politeknik Negeri Jakarta, Jurusan Teknik Elektro, Prodi Broadband Multimedia
Jl. Prof. Dr. G. A. Siwabessy, Kampus UI, Depok 16425
email: kh.arham@gmail.com

ABSTRACT
Attendance machines on the market mostly only use a standalone database system, which is 1 database for 1 machine despite applying the IoT principle. One protocol that is often used in IoT is the MQTT protocol. The MQTT protocol has often been studied, but it still utilizes public MQTT Servers that are ready to use and most are only done for 1-way communication. Public MQTT brokers must use the internet even if only for devices that are not too far away. Communication between clients and public brokers will be disrupted if the internet connection is interrupted, especially agencies in areas that have not been reached by a reliable ISP. To overcome this, it is necessary to implement MQTT using a local network that is already available without the need for internet. By implementing the MQTT protocol locally and two-way communication, PyQt and MongoDB as a centralized database management are expected to make it easier to monitor and manage the number of connected attendance machines. The system applied in this study has 3 components, namely MQTT Server, Monitor and Database Client, and Node Client. To conduct two-way communication each client publishes and subscribes to specific topics based on the client ID. Next will be investigated the effect of 3 parameters, i.e. the type of MQTT server used, the number of client nodes, and the process of updating the database to the delay in the authentication process. Obtained the difference in delay between the application of local and public MQTT servers is 816.75 ms. Increasing the number of client nodes also affects the delay of about 135.75 ms for each addition of 1 node client. The process of updating the MongoDB database in Python has the least delay effect compared to 2 other parameters, which is only 68.75 ms.

Key words: *Attendance; client monitor; client node; MongoDB; Mosquitto; MQTT; PyQt*

ABSTRAK
Mesin absensi di pasaran rata-rata hanya menggunakan sistem standalone database, yaitu 1 database untuk 1 mesin walaupun sudah menerapkan prinsip IoT. Salah satu protokol yang sering digunakan dalam IoT saat ini salah satunya protokol MQTT. Protokol MQTT sudah sering dijadikan pembahasan, namun masih memanfaatkan public MQTT Server yang sudah siap pakai dan kebanyakan hanya dilakukan untuk komunikasi 1 arah. Public MQTT broker harus menggunakan internet walaupun hanya untuk perangkat yang letaknya tidak terlalu jauh. Komunikasi antar client dengan cloud broker akan terganggu bila koneksi internet mengalami gangguan terutama instansi yang berada di daerah yang belum dijangkau oleh ISP yang handal. Untuk mengatasinya perlu menerapkan MQTT menggunakan jaringan lokal yang sudah tersedia tanpa memerlukan internet.

Dengan menerapkan protokol MQTT secara lokal dan komunikasi dua arah, PyQt serta MongoDB sebagai manajemen database terpusat diharapkan mempermudah dalam memonitor dan mengatur jumlah mesin absensi yang terkoneksi. Sistem yang diterapkan pada penelitian ini memiliki 3 komponen, yaitu MQTT Server, Monitor dan Database Client, dan Node Client. Untuk melakukan komunikasi dua arah setiap client melakukan publish dan subscribe ke topik tertentu berdasarkan client ID. Selanjutnya akan dicari tahu pengaruh 3 parameter, yaitu jenis server MQTT yang digunakan, jumlah client node, dan proses update database terhadap delay dalam proses autentikasi. Didapatkan perbedaan delay antara penerapan server MQTT lokal dan publik adalah 816.75 ms. Bertambahnya jumlah client node juga mempengaruhi delay sekitar 135.75 ms setiap penambahan 1 client node. Proses update database MongoDB pada Python memiliki pengaruh delay paling kecil dibandingkan 2 paramater lain, yaitu hanya 68.75 ms.

Kata kunci: *Attendance; client monitor; client node; MongoDB; Mosquitto; MQTT; PyQt*

PENDAHULUAN

Sistem absensi seringkali menjadi perhatian sebuah instansi dalam pengelolaan puluhan atau bahkan ratusan sumber daya manusia (karyawan), sehingga banyak perusahaan telah menerapkan sistem absensi digital. Kebanyakan instansi atau perusahaan memiliki gedung yang berdekatan atau hanya satu gedung, sehingga cukup meletakkan 1 atau 2 mesin absensi dengan pengumpulan data manual menggunakan *flashdisk* tanpa harus mengintegrasikannya ke jaringan lokal instansi tersebut. Beda halnya dengan instansi dengan area yang luas dan jarak antar gedung yang cukup jauh, sehingga mengharuskan instansi tersebut memasang mesin absensi di masing-masing gedung tempat karyawan bekerja yang jumlahnya tidak sedikit. Tentunya sangat merepotkan untuk mengumpulkan data di tempat-tempat yang berjauhan secara manual untuk masing-masing mesin. Kalaupun masing-masing gedung memiliki seorang perwakilan untuk mengumpulkan datanya, maka untuk menggabungkan semua data tersebut akan merepotkan bagian administrasi bila dilakukan setiap hari atau setiap minggu.

Untuk mesin yang memiliki fitur *logging* ke jaringan sendiri masih terbilang cukup mahal dan rata-rata memiliki database yang terpisah (*standalone*), yang beredar di pasaran misalnya ZT1200 atau ZT1100 yang menggunakan MySQL sebagai *standalone* database. Bagi instansi sejenis pabrik, manufaktur, atau bisnis dengan kemampuan *budget* tinggi, itu bukanlah masalah besar. Namun untuk instansi non-bisnis dengan kemampuan *budget* menengah atau instansi pada umumnya, dan ditambah dengan area luas, hal itu benar-benar harus dipertimbangkan. Oleh karena itu masih banyak yang menggunakan cara manual untuk perekapan data absensi.

Sekarang ini dengan adanya perkembangan sistem integrasi antar perangkat dengan jaringan seperti *Internet of Things* (IoT) ditambah dengan penerapannya yang tidak hanya pada komputer atau perangkat jaringan pada umumnya. Begitu mudahnya sekarang ini dalam menemukan *source code* atau *binary resource* untuk berbagai macam protokol pendukung IoT yang disediakan untuk para *hobby* dan *developer*, salah satunya protokol *Message Queiuing Telemetry Transport* (MQTT). MQTT menjadi salah satu protokol yang banyak digunakan pada aplikasi IoT karena memiliki ukuran paket data yang kecil sehingga cocok digunakan

pada aplikasi *real-time* atau aplikasi yang membutuhkan kemampuan *data streaming* yang baik. Dari beberapa info yang penulis dapatkan dari dosen dan referensi di internet, protokol MQTT sudah beberapa kali dijadikan bahan pembahasan untuk keperluan tugas akhir atau skripsi, namun masih memanfaatkan *public cloud* MQTT *broker* yang sudah siap pakai. Dalam penerapannya *cloud* MQTT *broker* harus menggunakan koneksi internet walaupun hanya untuk perangkat yang letaknya tidak jauh atau berdekatan. Tentu akan menghambat komunikasi antar *client* dengan *cloud broker* bila saat koneksi internet atau *Internet Service Provider* (ISP) yang sering mengalami gangguan terutama instansi yang berada di daerah tepencil atau masih belum dijangkau banyak ISP lain yang handal. Sehingga untuk mengatasinya agar lebih efisien perlu adanya penerapan MQTT yang cukup memanfaatkan jaringan lokal yang sudah tersedia di suatu instansi tanpa harus memakan *bandwidth* internet.

Salah satu penyedia *open-source* MQTT *client & broker* yaitu Eclipse Mosquitto. Dengan menggunakan Mosquitto kita dapat menerapkan protokol MQTT pada jaringan lokal. Mosquitto juga menyediakan *binary* yang mendukung berbagai macam sistem operasi seperti Windows, Linux, dan Mac. Bahkan Mosquitto juga dapat dipasang pada *single board computer* (SBC) seperti Raspberry Pi. Dengan memanfaatkan *low power single board computer* sebagai MQTT *broker* maka kita tidak perlu menggunakan sebuah *Personal Computer* (PC) yang harus *standby* selama koneksi dengan *broker* diperlukan terkait pertimbangan penggunaan daya. Selain itu output data dari *client* juga dapat ditampilkan pada sebuah database dengan merancang GUI pada SBC tersebut. Salah satu *platform* GUI yang dapat digunakan, yaitu Qt yang juga menyediakan *tools designer* untuk mempermudah *developer* saat merancang GUI dengan fitur konversi hasil desain ke bahasa pemrograman seperti C/C++ ataupun Python (PyQt).

Penerapan IoT dalam inovasi teknologi semakin menuntut adanya variasi data serta fleksibilitas dalam mengakses serta menyimpan data. Banyak perusahaan besar di dunia seperti Facebook yang telah bermigrasi dan menerapkan *database* NoSQL salah satunya MongoDB. Salah satu fitur MongoDB adalah menyimpan data dalam bentuk JSON *document.* Sebagai *programmer* pada umumnya berpikir secara objek, dengan cara ini MongoDB mendekati cara berpikir tersebut, sehingga lebih fleksibel dan handal dibandingkan *database* dengan pemodelan baris dan kolom seperti MySQL dan sejenisnya. Dalam akses pemrogramannya, Python sangat cocok dipasangkan dengan MongoDB karena dalam Python memiliki tipe data *dictionary* yang sangat mirip dengan JSON. Ditambah dengan adanya *library* atau *driver* yang tersedia pada Python untuk mengakses MongoDB, yaitu PyMongo. Dengan begitu akan mempermudah dalam implementasi dan menggabungkannya pada sistem *IoT.*

Tujuan dari penelitian ini adalah menerapkan protokol MQTT pada mesin absensi di *local area network* dengan mengandalkan *access point* terdekat untuk komunikasi lebih handal dan aman. Kemudian menerapkan algoritma komunikasi *bidirectional* dari protokol MQTT pada perangkat-perangkat yang terhubung pada jaringan lokal atau jaringan publik. Menggunakan aplikasi GUI *designer* PyQt5 dan penerapannya untuk menginput, mengedit, dan menampilkan data dari suatu *database* dengan konsep *NoSQL* salah satunya MongoDB dengan *library* PyMongo pada Python serta sebagai penerapan manajemen *database* secara terpusat. Terakhir adalah menganalisa pengaruh penggunaan server MQTT (lokal dan publik), proses *parsing data* pada Python dan *update* pada *database* MongoDB, serta penambahan jumlah *client* terhadap *delay* pada sistem.

METODE PENELITIAN

Alat yang akan dirancang berupa sistem absensi *multi-node* dengan database terpusat, yaitu terdapat banyak perangkat sebagai titik pengambilan data namun dengan tempat penyimpanan serta manajemen dan monitor terpusat dari satu perangkat saja. Sistem yang akan dirancang terdiri atas 3 komponen, yaitu *Local MQTT Server*, *Monitor* dan *Database Client*, dan *Node Client* pada satu jaringan lokal atau komunikasi tanpa melalui ISP atau jaringan eksternal.

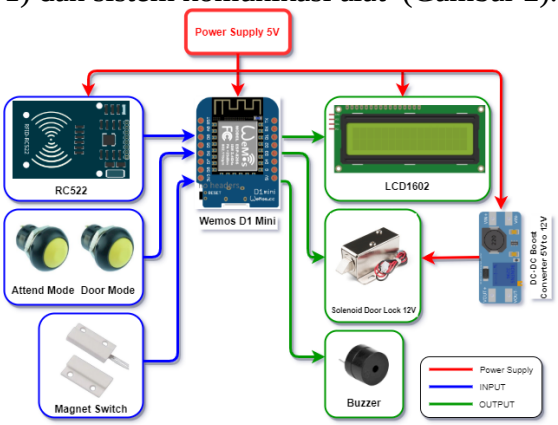
Local MQTT Server berfungsi sebagai Server MQTT yang digunakan antar *Client* MQTT untuk berkomunikasi atau tempat di mana fungsi *"publish"* dan *"subscribe"* akan diproses serta diatur oleh Server MQTT.

Monitor dan *Database Client* berfungsi untuk memonitor setiap *Node Client* yang terhubung dengan server MQTT yang sama serta merekam dan mengatur pengecekan data setiap *node* dan karyawan pada database. Pada *client* ini juga akan dijalankan aplikasi GUI yang telah dirancang dengan PyQt5 untuk membangun fungsi *monitor* dan sebagai jembatan penghubung dengan database.

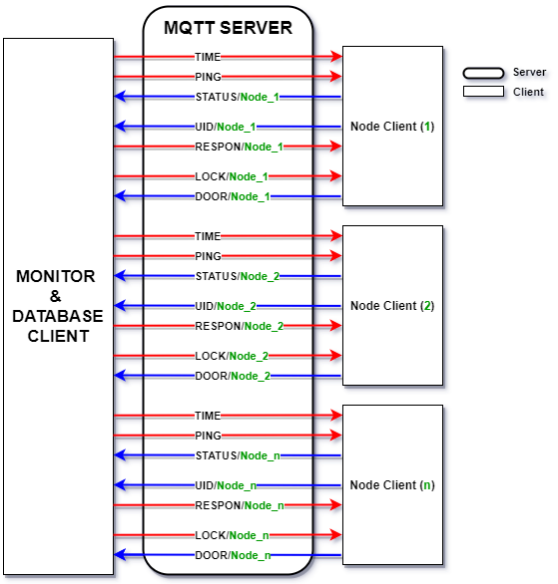
Node Client memiliki fungsi utama sebagai titik absensi. *Node Client* tidak dapat beroperasi bila *Client* Monitor dan Database tidak dijalankan walaupun *Node Client* sudah terkoneksi ke server. Fungsi opsional yaitu sebagai sistem pengunci pintu sekaligus membaca kondisi pintu terbuka atau tertutup. Kedua fungsi ini juga akan dimonitor oleh *Client* Monitor dan Database. Untuk mendukung kedua fungsi ini *user* dapat memilih apakah ingin menggunakan mode untuk membuka pintu (secara *default*) atau menggunakan mode presensi kehadiran.

Pada mode membuka pintu tidak akan dilakukan *update* data ke database, namun hanya terjadi proses autentikasi. Sedangkan untuk sistem kehadiran akan dilakukan *update* data ke database dan direkam berdasarkan pola 2 waktu, yaitu datang dan pulang per harinya untuk setiap *user*. Bagian kontroler *Node Client* akan diinstalasi di dalam ruangan, sedangkan bagian *interface* dan *scanner* berada di luar. Hal ini untuk mendukung fungsi *Node Client* sebagai sistem pengaman suatu ruangan. Namun yang terpenting *Node Client* harus dapat menjangkau *access point* terdekat.

Untuk mempermudah dalam memahami dan merealisasikan sistem, maka akan dibuat beberapa diagram blok di antaranya diagram blok hardware (Gambar 1) dan sistem komunikasi alat (Gambar 2).



Gambar 1. Diagram Blok Hardware



Gambar 2. Diagram Blok Sistem Komunikasi Alat

Sistem komunikasi terdiri dari MQTT *Server* dan *Client*. Setiap *client node* akan berkomunikasi dengan *client* monitor dan

database melalui server MQTT. Proses pengiriman data antar *client* dilakukan melalui topik yang telah dirancang sesuai Gambar 3.2. Dapat dilihat bahwa tanda panah biru merupakan topik yang di-*subscribe* oleh *client* monitor atau di-*publish* oleh *client node*. Sedangkan tanda panah merah merupakan topik yang di-*publish* oleh *client* monitor atau di-*subscribe* oleh *client node*. Arah tanda panah menunjukkan arah *publish* dari dan ke.

Terdapat 2 jenis topik dari sistem yang akan dibangun, yaitu *general topic* dan *specific topic*.

1. General Topic

Merupakan topik yang digunakan untuk mem-*publish* data yang berlaku untuk semua *client node*, tanpa membedakan *client ID*, dan tidak memiliki *sub-topic*, serta semua *client node* berlangganan ke topik tersebut. Hanya 2 *general topic* yang digunakan pada sistem ini, yaitu "PING" dan "TIME" yang akan di-*publish* dengan interval waktu tertentu. Dengan kata lain topik ini bersifat *broadcast*.

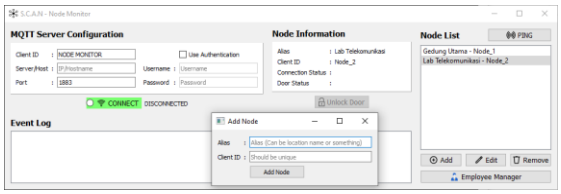
- **"PING"**: Digunakan untuk mengecek koneksi setiap *client node*.
- **"TIME"**: Digunakan untuk mengirim waktu ke setiap *client node*.

2. Specific Topic

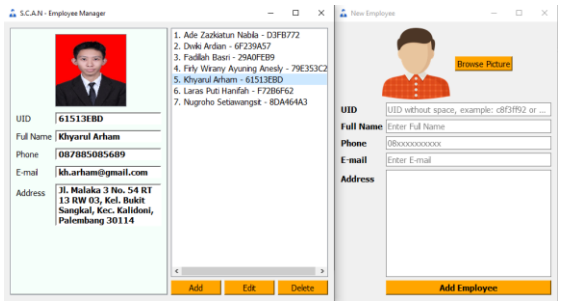
Merupakan topik yang hanya berlaku untuk *client node* tertentu berdasarkan *client ID*, topik memiliki karakter pemisah "/" dengan *sub-topic*, dan hanya *client node* dengan *client ID* bersangkutan yang melakukan *publish* ataupun *subscribe* ke topik tersebut. *Client ID* akan digunakan sebagai *sub-topic*. Pada perancangan sistem kali ini digunakan *client ID* dengan format "Node_n". Di bawah ini merupakan *specific topic* yang digunakan dalam sistem:

- **"STATUS/Node_n"**: *Client node* yang aktif akan mengirim respon ke topik ini setelah menerima data yang berasal dari topik "PING". *Client node* yang merespon dianggap aktif atau terkoneksi dengan server. Khusus pada topik ini masing-masing *client node* menggunakan fitur "*Last Will Message*" jika suatu waktu koneksi terputus. *Message* ini akan disimpan oleh server MQTT dan akan diteruskan ke *subscriber* topik ini yaitu *client* monitor jika salah satu *publisher* dalam hal ini *client node* bersangkutan tiba-tiba terputus koneksi dengan server.
- **"UID/Node_n"**: Digunakan dalam proses autentikasi oleh *client node* dengan cara mengirim data dengan format "MODE:KODE RFID TAG" ke *client* monitor setelah *user* melakukan *tapping*.
- **"RESPON/Node_n"**: Setelah *client* monitor mendapat data dari topik "UID/Node_n" *client* monitor akan melakukan proses autentikasi ke database. Hasil dari proses ini akan dikirim ke *client node* bersangkutan melalui topik "RESPON/Node_n".
- **"LOCK/Node_n"**: Digunakan oleh *client* monitor untuk membuka kunci (mengaktifkan solenoid) pada *client node* bersangkutan.
- **"DOOR/Node_n"**: Digunakan oleh *client node* untuk mengirim status pintu (terbuka atau tertutup) berdasarkan *state* yang terbaca oleh *magnetic switch*. *Client node* akan mengirim data pada topik ini hanya pada saat pertama kali menyala dan pada saat status pintu berubah (dari terbuka menjadi tertutup atau sebaliknya).

Berikutnya adalah merancang *layout* aplikasi yang dinamis. Ada 2 aplikasi yang akan dibuat, yaitu "*S.C.A.N – Node Monitor*" (Gambar 3) dan "*S.C.A.N – Employee Manager*" (Gambar 4).



Gambar 3. *S.C.A.N – Node Monitor*

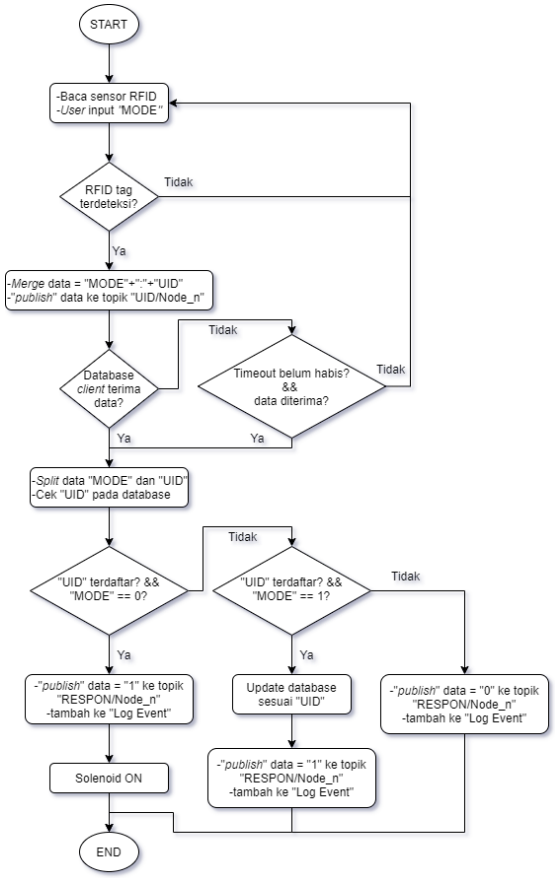


Gambar 4. S.C.A.N – Employee Manager

Aplikasi *Node Monitor* berfungsi untuk:

- Memonitor status koneksi dan kondisi pintu setiap *client node*.
- Merekam aktivitas *user* di setiap *client node* dengan cara menampilkannya ke *Event Log*.
- Mendukung fungsi *Create*, *Read*, *Update*, *Delete* (*CRUD*) pada *Node List* ke database MongoDB, serta mempermudah pengaturan koneksi ke *server* MQTT dengan adanya menu *MQTT Server Configuration*.

Sedangkan aplikasi *Employee Manager* berfungsi untuk mempermudah dalam manajemen dan membangun fungsi *CRUD* pada data *user* ke database MongoDB.



Gambar 5. *Flowchart* Sistem Autentikasi

Dari *flowchart* (Gambar 5) dapat dijabarkan proses autentikasi pada sistem sebagai berikut:

1. *User* menggunakan tombol pada mesin *client node* untuk memilih MODE 0 (membuka kunci pintu) atau MODE 1 (presensi).
2. Setelah *user* melakukan *tapping*, kode RFID (UID) dan mode akan di-*merge* menjadi satu dengan format "MODE:UID", lalu *client node* mem-*publish* data tesebut ke topik "UID/Node_n" dengan "Node_n" adalah *client ID* dari *client node* bersangkutan.
3. Bila data pada topik "UID/Node_n" belum diterima oleh *client* monitor selama waktu tertentu, maka *user* harus mengulang dari langkah 1.

4. Setelah diterima oleh *client* monitor data "MODE" dan "UID" akan dipecah lagi.
5. Jika UID terdaftar dan MODE = 0, maka *client* monitor akan mengirim karakter "1" pada topik "RESPON/Node_n" dan menambah data ke *Log Event*, sehingga pengunci pintu pada *client node* bersangkutan akan terbuka (Solenoid ON).
6. Jika UID terdaftar dan MODE = 1, maka *client* monitor akan meng-*update* tanggal dan jam pada *document* database dari *user* bersangkutan terlebih dahulu (disimpan dengan format **{"tanggal":["waktu hadir":"waktu pulang"]}**), baru mengirim karakter "1" pada topik "RESPON/Node_n" dan menambah data ke *Log Event*.
7. Jika UID tidak terdatar di database, *client* monitor akan langsung mengirim karakter "0" ke topik "RESPON/Node_n", sehingga *client node* akan memberitahu *user* pada LCD bahwa UID tidak dikenal atau tidak terdaftar. Terakhir pada *Log Event* akan muncul notifikasi UID tidak dikenal.

Setelah semua selesai dikonfigurasi dan dijalankan, tahap selanjutnya adalah proses *tapping* kartu RFID pada *Client Node*. Pengukuran dilakukan terhadap variasi berikut:

Tabel 1. Variasi Percobaan

Percobaan ke-	Server	Konfigurasi Jaringan	Jumlah *Client Node* Aktif	UID
1	Lokal (*localhost*)	I	1	Terdaftar
2	Lokal (*localhost*)	I	1	Tidak terdaftar
3	Lokal (*localhost*)	I	2	Terdaftar
4	Lokal (*localhost*)	I	2	Tidak terdaftar
5	Publik (test.mosquitto.org)	II	1	Terdaftar
6	Publik (test.mosquitto.org)	II	1	Tidak terdaftar
7	Publik (test.mosquitto.org)	II	2	Terdaftar
8	Publik (test.mosquitto.org)	II	2	Tidak terdaftar
9	Lokal (*localhost*) – Windows 10 PC	I	1	Tidak Terdaftar
10	Lokal (*localhost*) – Windows 10 PC	I	1	Terdaftar
11	Lokal (*localhost*) – Windows 10 PC	I	2	Tidak Terdaftar
12	Lokal (*localhost*) – Windows 10 PC	I	2	Terdaftar

Khusus UID terdaftar menggunakan “61513EBD” dengan nama pemilik “Khyarul Arham”, sedangkan UID yang tidak terdaftar pada *database* menggunakan “D3FB772” yang akan terdeteksi sebagai “*Unknown UID*”.

Variasi 1-4 Raspberry Pi difungsikan sebagai server dan tempat dijalankannya aplikasi *Node Monitor* dan *Employee Manager*. Sedangkan khusus variasi 9-12 server dan aplikasi dijalankan melalui PC dengan OS Windows 10 untuk melakukan perbandingan *delay* terhadap variasi 1-4. Sedangkan variasi 5-8 menggunakan server MQTT publik dengan aplikasi *Node Monitor* dan *Employee Manager* tetap dijalankan pada Raspberry Pi.

HASIL DAN PEMBAHASAN

Data pengukuran diambil dari “*Event Log*” pada aplikasi *Node Monitor* yang telah dirancang. Setiap percobaan dilakukan sebanyak 17 kali dengan variasi berdasarkan Tabel 1. Berikut tabel hasil pengukuran berdasarkan tabel variasi percobaan.

Tabel 2. Hasil Pengukuran *Delay* (ms) Proses Autentikasi (Raspberry Pi)

No.	Variasi Percobaan ke-							
	1	2	3	4	5	6	7	8
1	202	171	231	422	1415	862	1041	1142
2	210	101	322	322	944	992	1092	1121
3	182	90	321	281	912	823	1063	1011
4	231	232	221	231	1051	942	1182	983
5	213	193	241	244	942	861	1122	1101
6	171	111	251	321	1041	941	1843	1012
7	131	102	331	271	864	1032	1113	1101
8	121	124	352	243	1042	821	963	1015
9	151	212	271	181	923	914	1112	1082
10	132	101	221	191	871	933	1183	1131
11	332	172	221	181	911	841	1020	1023
12	332	101	272	192	982	1031	1060	1012
13	151	100	231	612	902	981	1071	1001
14	182	191	371	160	914	863	1052	1161
15	181	111	361	141	1051	1073	1213	1073
16	132	92	271	221	1042	742	1044	991
17	140	122	291	182	1041	830	1812	962
Rata-rata *delay*	188	137	281	259	991	911	1176	1054

Tabel 3. Hasil Pengukuran *Delay* (ms) Proses Autentikasi (Windows 10 PC)

No.	Variasi Percobaan ke-			
	9	10	11	12
1	51	93	180	394
2	94	127	186	161
3	101	76	154	209
4	126	122	204	238
5	54	96	120	152
6	117	95	140	381
7	89	95	102	230
8	174	117	231	191
9	81	126	90	177
10	64	99	272	128
11	320	73	179	220
12	57	178	147	180
13	108	145	195	337
14	58	96	92	167
15	134	92	184	178
16	49	93	242	90
17	66	73	236	147
Rata-rata *delay*	103	106	174	211

1. Pengaruh Proses *Update Database*

Tahap pertama adalah mencari tahu selisih *delay* pada saat menggunakan server dan jumlah *client node* aktif yang sama, namun hanya jenis UID yang berbeda, yaitu terdaftar dan tidak terdaftar pada database. Selisih ini dapat dihitung pada percobaan variasi antara:

- 1 dan 2: selisih rata-rata delay = 188-137 = 51 ms
- 3 dan 4: selisih rata-rata delay = 281-259 = 22 ms
- 5 dan 6: selisih rata-rata delay = 991-911 = 80 ms
- 7 dan 8: selisih rata-rata delay = 1176-1054 = 122 ms

Pada variasi 1, 3, 5, dan 7 dari tabel hasil pengukuran diketahui selalu memiliki rata-rata *delay* yang lebih besar dibandingkan dengan variasi 2, 4, 6, dan 8. Hal ini dikarenakan variasi 1, 3, 5, dan 7 menggunakan UID yang terdaftar pada database, sedangkan variasi 2, 4, 6, dan 8 menggunakan UID yang tidak terdaftar pada *database*. UID yang terdaftar akan melalui proses *query* dan *update database* MongoDB dalam program Python, sedangkan UID yang tidak terdaftar hanya melalui proses *query* tanpa perlu dilakukan *update* data pada *database* MongoDB. Jika UID yang bersangkutan tidak *match* (tidak terdaftar), aplikasi *Node Monitor* akan langsung mem-*publish* karakter "0" pada topik "RESPON/clientID". Dari selisih rata-rata *delay* di atas, maka didapat waktu yang diperlukan oleh MongoDB dengan Python pada sistem ini untuk meng-update database:

$$Rata - rata\ Waktu\ Update\ Database = \frac{51 + 22 + 80 + 122}{4} = 68.75\ ms$$

Perintah *update database* pada kode program Python untuk aplikasi *Node Monitor*, yaitu "employeeCollection.update_one(query, newValue)". Kode ini dipanggil hanya ketika UID yang diterima oleh aplikasi *Node Monitor* cocok atau terdapat pada *database* (UID terdaftar pada *database*). Respon bahwa UID terdaftar dan autentikasi sukses ditandai dengan dijalankannya perintah "client.publish('RESPON/' + clientID, name)" untuk mengirim nama dari pemilik UID tersebut.

2. Pengaruh Jumlah *Client Node* yang Aktif

Tahap selanjutnya adalah mencari tahu selisih *delay* pada saat menggunakan server

dan tipe UID (antara terdaftar atau tidak terdaftar) yang sama, namun hanya jumlah *client node* aktif saja yang berbeda. Selisih ini dapat dihitung pada percobaan variasi antara:

- 3 dan 1: selisih rata-rata delay = 281-188 = 93 ms
- 4 dan 2: selisih rata-rata delay = 259-137 = 122 ms
- 7 dan 5: selisih rata-rata delay = 1176-281 = 185 ms
- 8 dan 6: selisih rata-rata delay = 1054-911 = 143 ms

Pada variasi 3, 4, 7, dan 8 dari tabel hasil pengukuran diketahui selalu memiliki rata-rata *delay* yang lebih besar dibandingkan dengan variasi 1, 2, 5, dan 6. Hal ini dikarenakan variasi 1, 2, 5, dan 6 aplikasi *Node Monitor* hanya berkomunikasi dengan hanya 1 *client node aktif*, sedangkan variasi 3, 4, 7, dan 8 aplikasi *Node Monitor* berkomunikasi dengan 2 *client node* aktif. Semakin banyak *client node* maka aplikasi harus membuat lebih banyak *thread* untuk meng-*handle* masing-masing *client node*. Dari selisih rata-rata *delay* di atas, maka didapat pengaruh jumlah *client node* aktif terhadap waktu yang diperlukan saat proses autentikasi pada sistem ini:

$$Delay/penambahan\ client\ node = \frac{93 + 122 + 185 + 143}{4}$$
$$= 135.75\ ms/\text{penambahan } client\ node$$

Namun hasil *delay* dari pengaruh penambahan jumlah *client node* aktif ini belum diketahui apakah akan meningkat secara linier atau non-linier. Untuk mengetahuinya harus dilakukan pengujian dengan lebih dari 2 *client node*.

3. Pengaruh Server yang Digunakan

Tahap berikutnya adalah mencari tahu *delay* dengan cara membandingkan 2 variasi saat beroperasi dengan jumlah client node dan tipe UID (antara terdaftar atau tidak terdaftar) yang sama, namun menggunakan server MQTT yang berbeda (antara server lokal hasil rancangan dan server publik). Selisih ini dapat dihitung pada percobaan variasi antara:

- 5 dan 1: selisih rata-rata delay = 991-188 = 803 ms
- 6 dan 2: selisih rata-rata delay = 911-137 = 774 ms
- 7 dan 3: selisih rata-rata delay = 1176-281 = 895 ms
- 8 dan 4: selisih rata-rata delay = 1054-259 = 795 ms

Pada variasi 5, 6, 7, dan 8 hasil pengukuran *delay* lebih besar dibandingkan dengan variasi 1, 2, 3, dan 4. Ini disebabkan variasi 5, 6, 7, dan 8 menggunakan server MQTT publik dan membutuhkan koneksi internet untuk mengirim dan menerima data. Sedangkan pada variasi 1, 2, 3, dan 4 sudah menggunakan server MQTT lokal yang telah dirancang, sehingga pengiriman data tidak melalui internet dan tentunya akan menekan waktu yang diperlukan dalam proses autentikasi. Dengan begitu dapat dihitung *delay* antara penerapan server MQTT lokal dengan server MQTT publik pada sistem ini:

$$Delay_{MQTT\ publik - MQTT\ lokal} = \frac{803 + 774 + 895 + 795}{4}$$
$$= 816.75\ ms$$

4. Pengaruh Perbedaan Spesifikasi Perangkat

Pada bagian ini dikhususkan untuk percobaan variasi 9-12 yang menggunakan PC dengan OS Windows 10. Berdasarkan tabel 4.3 bila dibedakan dengan percobaan variasi 1-4, maka dapat dibandingkan nilainya:

- 2 dan 9: selisih rata-rata delay = 137-103 = 34 ms
- 1 dan 10: selisih rata-rata delay = 188-106 = 82 ms
- 4 dan 11: selisih rata-rata delay = 259-174 = 85 ms

- 3 dan 12: selisih rata-rata delay = 281-211 = 70 ms

Pada variasi 1-4 hasil pengukuran *delay* selalu lebih besar dibandingkan percobaan variasi 9-12 yang menggunakan PC sebagai server dan dijalankannya aplikasi *Node Monitor* dan *Employee Manager*. Dengan begitu dapat diketahui selisih *delay* antara penerapan PC dan mini PC:

$$Delay_{Raspi-PC} = \frac{34 + 82 + 85 + 70}{4} = 67.75\ ms$$

Dari hasil di atas ada beberapa faktor lain yang juga memungkinkan dapat mempengaruhi hasil pengukuran di antaranya *resource* dan spesifikasi pada *hardware* itu sendiri. Misalnya pada variasi percobaan ke-4 no. 13 terukur lonjakan *delay* cukup besar, yaitu 612 ms, penyebab spesifik belum bisa diketahui, namun diduga bisa terjadi karena spesifikasi RAM yang rendah pada server (Raspberry Pi 3B): DDR2 900MHz dengan kapasitas yang kecil, yaitu 1 GB. Berbeda dengan komputer pada umumnya yang menggunakan DDR3 yang tentunya beberapa kali lipat lebih cepat dibanding RAM tipe DDR2. Ditambah lagi pada aplikasi S.C.A.N – *Node Monitor* yang telah dirancang menerapkan fitur *multithread* untuk meng-*handle* banyak *client node* yang tentunya akan membutuhkan banyak alokasi memori RAM. Hal yang paling berpengaruh terhadap lonjakan *delay*, yaitu banyaknya *thread* yang dijalankan pada mesin *node client* sendiri. Sebab ESP8266 sendiri memiliki kecepatan dan memori yang sangat kecil.

KESIMPULAN

Dari percobaan dan pengukuran serta analisa yang telah dilakukan maka dapat ditarik kesimpulan sebagai berikut:

1. Penggunaan server MQTT lokal dapat mengurangi *delay* proses pengiriman data dibandingkan server MQTT publik. Pada proses autentikasi UID dalam sistem ini penggunaan server MQTT lokal memiliki waktu rata-rata 816.75 ms lebih cepat dibandingkan server MQTT publik.
2. Bertambahnya jumlah *client node* yang aktif akan membuat bertambahnya *delay* saat proses autentikasi, apakah itu menggunakan server lokal maupun server publik. Dari pengujian didapatkan peningkatan *delay*, yaitu 135.75 ms setiap penambahan 1 *client node*.
3. Proses *update database* memiliki pengaruh paling kecil terhadap *delay* dalam proses autentikasi dibandingkan pengaruh penggunaan server MQTT (antara lokal atau publik) ataupun pengaruh bertambahnya jumlah *client node*. Pada sistem ini didapat lamanya waktu rata-rata yang diperlukan untuk meng-*update* data dengan MongoDB adalah hanya 68.75 ms.
4. Perbedaan perangkat yang digunakan mempengaruhi performa saat proses autentikasi. Dari pengujian antara Raspberry Pi Ubuntu Mate 18.04 dan PC Windows 10 didapatkan perbedaan *delay*, yaitu 67.75 ms.

DAFTAR PUSTAKA

[1]Antonić, A., M. Marjanović, P. Skočir, and I. P. Žarko. (2015). Comparison of the CUPUS Middleware and MQTT Protocol for Smart City Services. In 2015 13th International Conference on Telecommunications (ConTEL), 1–8. doi:10.1109/ConTEL.2015.7231225.

[2]Atmoko, Rachmad Andri. (2019). Dasar Implementasi Protokol MQTT

Menggunakan Python dan NodeMCU. Blitar: Mokosoft Media.

[3]Bellavista, Paolo, Carlo Giannelli, and Riccardo Zamagna. (2017). The Pervasive Environment Sensing and Sharing Solution. Sustainability 9 (4): 585. doi:10.3390/su9040585.

[4]Fremantle, P., B. Aziz, J. Kopecký, and P. Scott. (2014). Federated Identity and Access Management for the Internet of Things. In 2014 International Workshop on Secure Internet of Things, 10–17. doi:10.1109/SIoT.2014.8.

[5]IdCloudHost. (2017, Juli 7). Mengenal Lebih Dekat Tentang Database MongoDB. Januari 23, 2019. https://idcloudhost.com/mengenal-lebih-dekat-tentang-database-mongodb/

[6]Light, R.A. (2017). Mosquitto: server and client implementation of the MQTT protocol. Journal of Open Source Software, 2(13), 265, doi:10.21105/joss.00265.

[7]R. P Padhy, M. R. Patra, S. C. Satapathy. (2011). RDBMS to NoSQL: Reviewing Some Next-Generation Non-Relational Database's. International Journal of Advance Engineering Sciences and Technologies. Vol. 11, Issue No. 1, 015-030.

[8]Riverbank Computing Limited. What is Pyqt?. Januari 23, 2019. https://www.riverbankcomputing.com/software/pyqt/intro

[9]Sasaki, Y., Yokotani, T., & Mukai, H. (2019). MQTT over VLAN for Reduction of Overhead on Information Discovery. 2019 International Conference on Information Networking (ICOIN). doi:10.1109/icoin.2019.8718125.

[10]Schulz, M., F. Chen, and L. Payne. (2014). Real-Time Animation of Equipment in a Remote Laboratory. In 2014 11th International Conference on Remote Engineering and Virtual Instrumentation (REV), 172–76. doi:10.1109/REV.2014.6784247.

[11]Sharma, T., & Aarthy, S. L. (2016). An automatic attendance monitoring system using RFID and IOT using Cloud. 2016 Online International Conference on Green Engineering and Technologies (IC-GET). doi:10.1109/get.2016.7916851.

[12]Thangavel, D., X. Ma, A. Valera, H. X. Tan, and C. K. Y. Tan. (2014). Performance Evaluation of MQTT and CoAP via a Common Middleware. In 2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 1–6. doi:10.1109/ISSNIP.2014.6827678.