# SISTEM IOT BERBASIS PROTOKOL MQTT DENGAN MIKROKONTROLER ESP8266 DAN ESP32

**Ida Bagus Putu Widja**[1*]
Program Studi Sistem Komputer, STMIK Stikom Bali
Jalan Raya Puputan No.86 Renon Denpasar Bali, (0361) 244445
[*]Email: ibpwidja@gmail.com

**Abstrak**

*Internet of Things (IoT) adalah merupakan salah satu elemen dari Revolusi Industri 4.0 yang mengubah cara interaksi M2M (Machine to Machine) yang saling terkoneksi melalui jaringan Internet. IoT berdampak cukup luas dan dapat diterapkan ke berbagai bidang yang memerlukan kontrol interaksi M2M secara otomatis. Makalah ini berisi prototype sistem IoT menggunakan jaringan nirkabel (wifi) yang menghubungkan node-node dimana setiap node berisi sensor atau aktuator yang dikendalikan oleh mikrokontroler ESP8266 dan ESP32. Metode penelitian berdasarkan atas tahapan model penelitian rancang bangun untuk membentuk sistem dan alur kerja hardware dan software pada sistem IoT. Protokol yang dipilih adalah Message Queue Telemetry Transport (MQTT) yang merupakan protokol dengan bandwidth minimum, murah dan ringan sehingga mudah bagi client (node) untuk melakukan Publish/Subscribe pada topik tertentu di host yang disebut dengan Broker. Pengujian dilakukan dengan melihat unjuk kerja sensor dan aktuator serta interaksi antar client. Meskipun ada ganguan koneksi pada wifi router, interaksi antar client terjadi cukup baik dan hasil pengujian secara keseluruhan menunjukkan sistem telah bekerja sesuai harapan.*

***Kata kunci**: ESP32; ESP8266; IoT; Mosquitto; MQTT*

## 1. PENDAHULUAN

*Internet of Things* (*IoT*) adalah suatu teknologi interaksi *Machine To Machine* (*M2M*) melalui jaringan internet tanpa perantara manusia. *IoT* memungkinkan perangkat (*things*) di setiap titik *client* dapat saling berinteraksi lewat internet dan dapat dimonitor ataupun dikendalikan dari manapun.

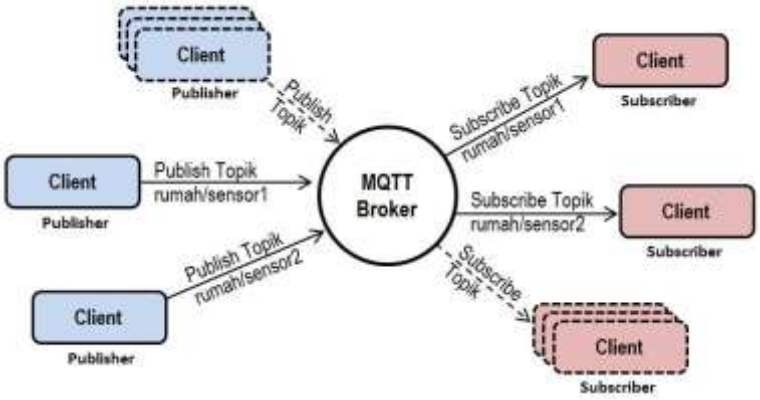
Pilihan protokol komunikasi di internet yang populer adalah *HTTP* yang berjalan diatas *TCP/IP* berbasis arsitektur *client/server*. Akan tetapi protokol tersebut tidak dioptimasi untuk rancangan sistem *embedded* mikrokontroler yang hanya memiliki kemampuan hardware (CPU, memori) yang sangat terbatas. Perangkat *IoT* biasanya menggunakan sensor-sensor untuk pengindraan lingkungan yang diletakkan di posisi tertentu sehingga memerlukan pengontrol kecil (mikrokontroler) dengan fleksibilitas tinggi, misalnya modul ESP8266 atau ESP32. Tiap-tiap sensor akan terhubung ke modul mikrokontroler yang berlaku sebagai *client* pada sistem, jadi perlu dicarikan solusi alternatif protokol lainnya yang lebih ringan dan sederhana daripada *HTTP*. (Valerie, 2012)

*MQTT* (*Message Queuing Telemetry Transport*) adalah salah satu protokol pilihan untuk perangkat *IoT* yang dapat berjalan pada jaringan bandwith rendah (*latensi* tinggi) sehingga ideal digunakan untuk koneksi *M2M* sistem *embedded*. Protokol *MQTT* sudah ada sejak tahun 1999 oleh *IBM* akan tetapi baru berkembang sangat pesat belakangan ini di tahun era *IoT* (Michael Yuan, 2017). *MTTQ* berbasis data-senteris sedangkan *HTTP* berbasis dokumen-senteris, keuntungan utamanya adalah sangat ringan (data ditransfer sebagai *array byte*) dengan model arsitektur *publish/subscribe* sehingga membuat protokol *MQTT* sangat cocok diterapkan pada perangkat bersumber daya terbatas.

Berdasarkan latar belakang yang sudah diuraikan diatas, maka yang menjadi pokok permasalahan adalah bagaimana merancang dan merealisasikan teknologi *Intenet Of Things* menggunakan protokol *MQTT* yang dapat mengatur data dari sensor/aktuator pada modul mikrokontroler ESP8266 dan ESP32 yang terkumpul pada *MQTT Broker* untuk menghasilkan sistem *embedded* yang terkoneksi ke jaringan internet melalui *wifi router*.

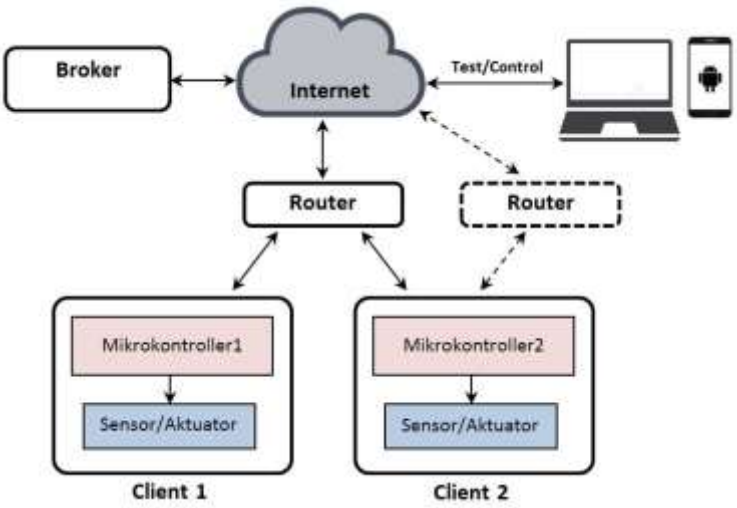
## 2. METODOLOGI

*MQTT* menggunakan model *publish/subscribe* yang memerlukan kontrol terpusat yang disebut dengan *Broker* seperti pada Gambar 1. *Client* yang berperan sebagai *publisher* akan mem-*publish* topik tertentu ke *Broker* misalnya topik "rumah/sensor1". Kemudian ada *client* lainnya yang berperan sebagai *subscriber* yang akan men-*subscribe* topik tertentu pada *Broker*, sistem dapat memiliki banyak *subscriber* yang men-*subscribe* individual topik tertentu misalnya ada topik "rumah/sensor1" atau "rumah/sensor2" dimana Topik tersebut di *publish* oleh lebih dari satu *client publisher*. *Broker* memiliki tugas untuk menampung sementara topik dari beberapa *publisher* dan meneruskan pesan tersebut ke banyak *subscribernya*.




**Gambar 1. Skema Komunikasi MQTT**

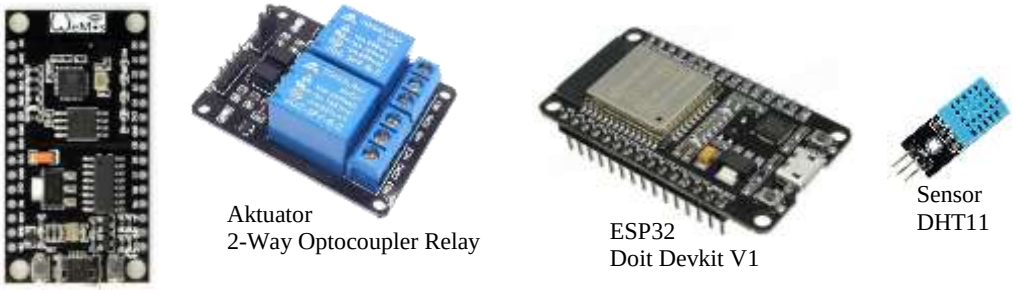
*Publisher* model *MQTT* tidak memiliki koneksi langsung ke *subscriber* begitu juga sebaliknya, mereka hanya dihubungkan oleh *Broker*. *Subscriber* dapat memilih topik yang tersedia pada *Broker*, layaknya memilih saluran (*channel*) televisi yang mana penerima siaran TV tidak berhubungan langsung dengan siaran acaranya. Topik yang di-*publish* oleh *client publisher* hanya akan diteruskan atau di-*broadcast* oleh *Broker* ke *client subscriber*-nya atau dengan kata lain *publisher* topik dapat saja tidak mengetahui siapa saja *subscriber*-nya. *Client* dapat juga sekaligus menjadi *publisher* dan *subscriber* pada topik yang sama ataupun berbeda.




**Gambar 2. Blok Diagram Sistem**

Rancangan sistem *IoT* ini menggunakan 2 buah *client* dengan mikrokontroler yang berbeda termasuk sensor atau aktuatornya (Gambar 2.). Setiap *client* dapat terkoneksi dengan

*wifi router* yang sama ataupun berbeda (gambar dengan garis putus-putus) menuju ke *MQTT Broker* melalui jaringan internet. Laptop atau *smartphone* (yang juga merupakan *client MQTT*) digunakan untuk melakukan test dan kontrol pada sistem yang terkoneksi ke *Broker*. *Client* akan melakukan *publish/subscribe* pada topik tertentu yang merupakan simpul penghubung antara *publisher* dan *subscriber*. Topik pada sistem *MQTT* tidak memiliki struktur formal sehingga *publisher* bebas memilih strukturnya topiknya sendiri. *Client1* akan menggunakan ESP8266 *NodeMCU V3* dari *Wemos* dengan aktuator *2-Way Relay* sebagai *switch* sedangkan *client2* menggunakan ESP32 *Devkit Vi* dari *Doit* dengan sensor suhu DHT11 seperti pada Gambar 3.



ESP8266 Wemos
NodeMCU V3

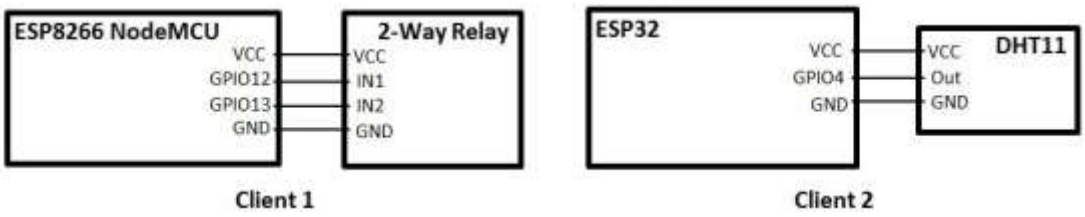
Aktuator
2-Way Optocoupler Relay

ESP32
Doit Devkit V1

Sensor
DHT11

**Gambar 3. Modul Mikrokontroler dan Sensor/Aktuator**

Skema rangkaian Gambar 4. memperlihatkan koneksi antara pin pada mikrokontroler dan pin pada aktuator dimana 2-Way Relay akan menerima perintah dari ESP8266 (pin GPIO12 dan GPIO13) untuk menutup 2 buah *switch* pada *relay*. Data keluaran suhu dan kelembaban udara pada sensor DHT11 akan diterima oleh ESP32 melalui pin GPIO4. Sensor/aktuator menerima catu daya dari mikrokontroler dengan tegangan 3.3Volt.



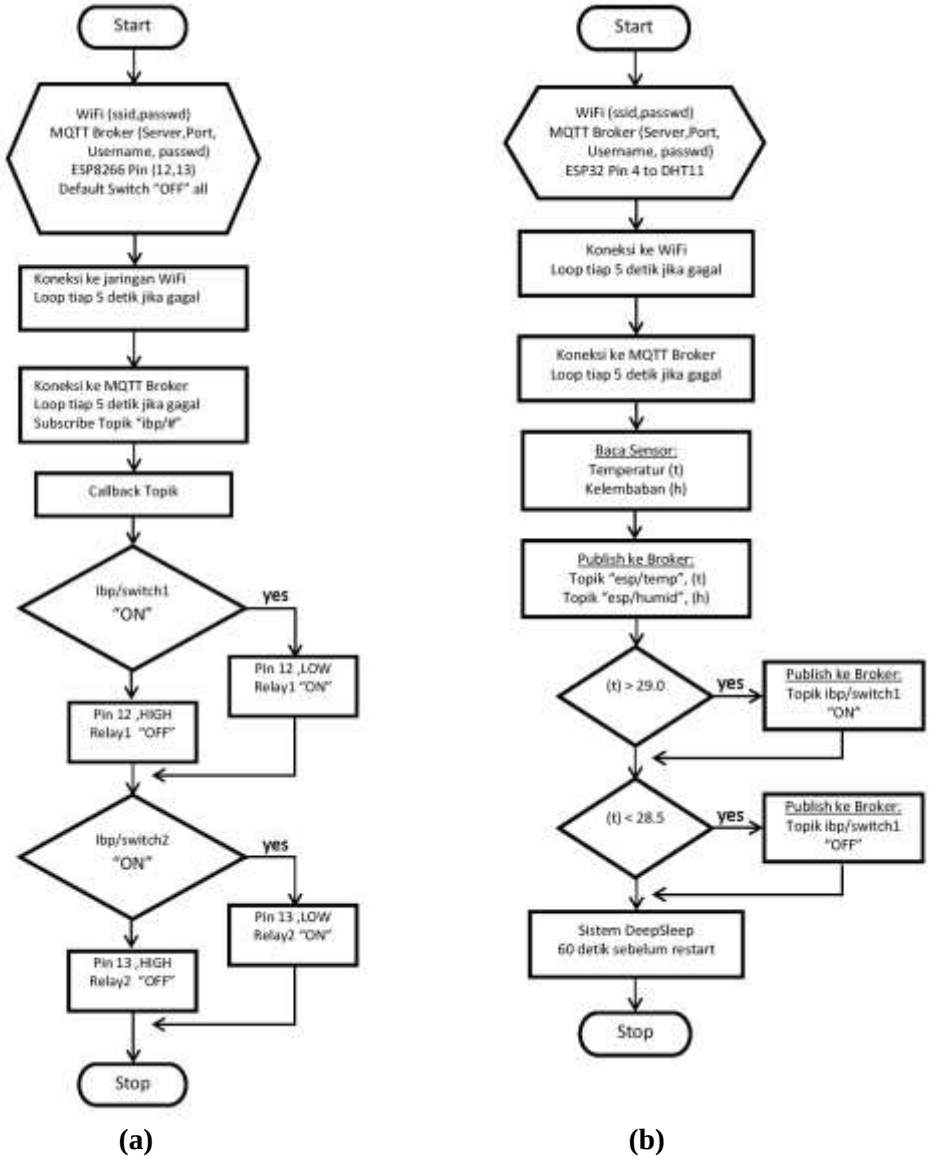

**Gambar 4. Skema Rangkaian Client**

Program yang ditanamkan pada kedua mikrokontroler menggunakan aplikasi *IDE* (*Integrated Development Environment*) *Arduino* yang mana saat memprogram tidak memerlukan rangkaian tambahan lainnya karena mikrokontroler sudah mengandung modul program terintegrasi dengan konektor *micro-USB*. Gambar 5a. dan Gambar 5b. adalah alur program secara garis besar dimana ESP8266 dirancang sebagai *Client Subscriber* dan ESP32 sebagai *Client Publisher*.

Program diawali dengan deklarasi, kemudian sistem mencoba untuk koneksi ke jaringan *wifi*, jika berhasil sistem akan lanjut mencoba koneksi ke *MQTT Broker* (Gambar 5a-b*).* Jika koneksi ke *Broker* berhasil ESP8266 akan men-*subscribe* topik "ibp/switch1" dan "ibp/switch2" sekaligus dengan notasi "ibp/#" saja. Rutin *callback* topik "ibp/switch#" akan menerima data dari *Broker* yang isi datanya hanya *string* "ON" dan "OFF". Pin 12 (*state Low*) akan mengaktifkan *Relay1* (*switch1)* sedangkan Pin 13 (*state Low*) mengaktifkan *Relay2* (*switch2*) jika pesan "ibp/switch# = "ON".

Modul ESP32 akan membaca data pada sensor DHT11 yaitu temperatur (t) dan kelembaban udara (h), kemudian mem-*publish* pada topik "esp/temp" dan "esp/humid". Jika temperatur > 29$^{o}$C sistem akan mem-*publish* topik "ibp/switch1" = "ON" dan menjadi

"OFF" jika temperatur < 28.5ºC. Atau dengan kata lain *client2* (ESP32) akan menyalakan *Relay1* pada *client1* (ESP8266) jika temperatur di lokasi *clien2* lebih dari 29ºC. Sistem memiliki mekanisme untuk menghemat daya listrik dengan mengaktifkan fungsi *Deepsleep* selama 1 menit sebelum *restart* kembali untuk mem-*publish* data berikutnya setiap 1 menit.




**(a)** **(b)**

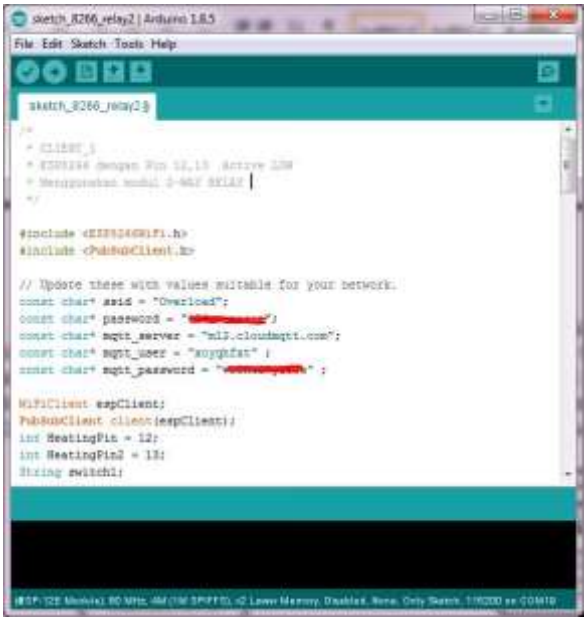
**Gambar 5. Algoritma Program Mikrokontroler (a) Algoritma Mikrokontroler ESP8266, (b) Algoritma Mikrokontroler ESP32**

## 3. HASIL DAN PEMBAHASAN

Realisasi rancangan sistem *IoT* pada kedua sisi *client* ditampilkan pada Gambar 6. mengikuti skema rangkaian Gambar 4. Catu daya sistem dapat melalui konektor *micro-USB* . Konektor tersebut dapat dihubungkan dengan *charger smartphone* umum, bisa dengan *powerbank* atau dapat juga memakai baterai 9V dengan regulator supaya keluaran catunya mencapai 5V.

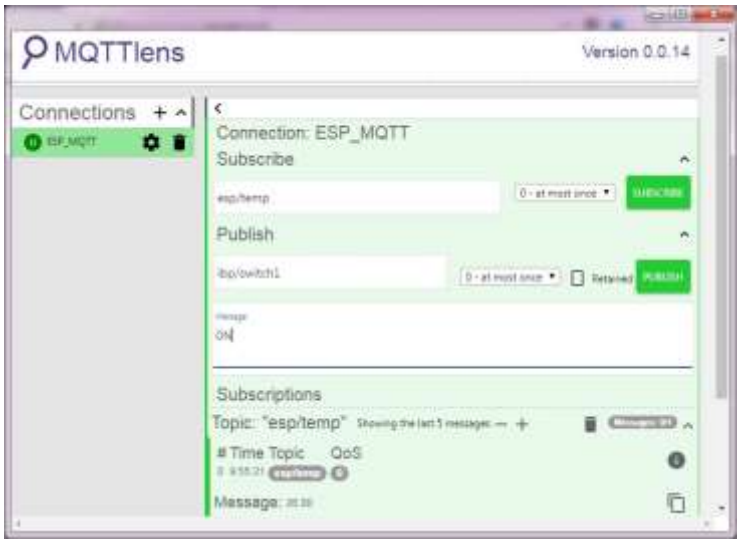

**Gambar 6. Realisasi Rangkaian**

Sistem *IoT* pada sisi *Broker* menggunakan layanan jasa *MQTT Broker online* berbasis *Mosquitto* di situs *www.cloudmqtt.com* dengan layanan *server m12.cloudmqtt.com*. Sistem pada awalnya disiapkan menggunakan lokal *Broker* modul komputer *Raspberry Pi* dengan aplikasi *Mosquitto*, akan tetapi karena tidak punya akses ke *admin* Router maka sistem tidak dapat diakses dari lingkungan luar jaringan lokal. Oleh karena itu sistem akhirnya dipilih menggunakan layanan *Broker online*. Dari registrasi di situs *cloudmqtt.com* maka akan langsung mendapatkan informasi *nama_server*, *port*, *username* dan *password* untuk diterapkan pada *client MQTT*.




**Gambar 7. Tampilan Arduino IDE**

Gambar 7. adalah contoh tampilan aplikasi Arduino IDE yang dipasang pada laptop untuk keperluan *upload* program ke mikrokontroler. *Arduino IDE* merupakan program *open source* yang cukup populer yang dibuat untuk modul mikrokontroler *Arduino* yang dapat diunduh gratis di situs *www.arduino.cc*. Untuk keperluan pengujian sistem, dibutuhkan aplikasi tambahan untuk di pasang di laptop. Aplikasi ini memiliki kemampuan untuk melakukan *publish/subscribe* secara mandiri sehingga unjuk kerja sistem dapat dengan mudah dievaluasi, aplikasi tersebut adalah *MQTTLens* (Gambar 8.) merupakan bagian dari *Google Chrome*.




**Gambar 8. Tampilan MQTTLens**

### 3.1. Hasil Pengujian

Rangkaian skenario pengujian ditampilkan pada Tabel 1., dimana tahapan pengujian yang dilakukan adalah sebagai berikut:

1) Uji sederhana kerja Sensor → memastikan Sensor dalam keadaan normal.
2) Uji sederhaha kerja Aktuator →memastikan *2-Way Relay* dalam keadaan normal.
3) Uji koneksi ke *wifi* dan ganguannya →memastikan fungsi *reconnect* dapat bekerja.
4) Uji Fungsional →memastikan unjuk kerja sistem sesuai harapan.
   a) Interaksi sistem dengan *MQTTLens* →memastikan interaksi antar *client* terjadi melalui proses *publish* dan *subscribe* pada topik yang telah ditetapkan.
   b) Interaksi antar *client* →memastikan perubahan temperatur sensor (ESP32) dapat mengaktifkan aktuator (ESP8266).

**Tabel 1. Skenario Pengujian dan Hasil**

| No | Skenario | Pengamatan | Hasil | Keterangan |
|---|---|---|---|---|
| 1 | ESP32 di-*upload* dengan program ringkas untuk data dari DHT11. Kemudian secara bertahap didekatkan ke pemanas. | Sensor dapat menunjukkan data suhu dan kelembaban udara. Ada peningkatan data suhu dan pengurangan kelembaban saat Sensor dekat pemanas. | Sensor normal | Batasan: Tidak melihat tingkat akurasi data DHT11 |
| 2. | Bagian kontak *output* 2-*Way Relay* dipoin dengan Multimeter. Aktuator diberi catu daya kemudian pin IN1& IN2 di *Ground* untuk mengaktifkan relay. | Kontak *2-Way Relay* aktif (pada posisi pin IN1 dan IN2 di *Ground*) dan Multimeter menunjukkan koneksi | Aktuator normal | |
| 3. | *Client* diaktifkan dengan program utuh, kemudian wifi router di "OFF" dan di "ON" lagi untuk skema uji gangguan jaringan. | Pengamatan dilakukan pada *Port Serial Arduino IDE baud rate* 115200. *client* berhasil terkoneksi ke wifi Router dan berhasil melakukan *reconnect* saat mengalami gangguan. | *Client* berhasil *connet* ke wifi & melakukan *reconnect* saat ada gangguan. | IP *Client* 1: 10.10.10.196<br>IP *Client* 2: 10.10.10.224<br>IP Router: 10.10.10.1 |

| No | Skenario | Pengamatan | Hasil | Keterangan |
|---|---|---|---|---|
| 4. a | Kedua *client* diaktifkan, kemudian *MQTTLens* pada laptop dikoneksikan ke Broker m12.cloudmqtt.com.<br><br>*Publish* pada Topik: "ibp/switch1" dan "ibp/switch2" dengan message "ON" kemudian "OFF"<br>*Subscribe* pada Topik: "esp/temp" dan "esp/humid"<br><br>Koneksi laptop ke internet menggunakan 2 cara dengan WiFi Router dan kemudian dengan GSM (XL) | Publish pada Topik "ibp/switch1" = "ON" Relay1 client ESP8266 hidup.<br>"ibp/switch1" = "OFF" Relay1 client ESP8266 mati.<br>"ibp/switch2" = "ON" Relay2 client ESP8266 hidup.<br>"ibp/switch2" = "OFF" Relay2 client ESP8266 mati.<br><br>*Subscribe* pada Topik:<br>"esp/temp" → data suhu ditampilkan di *MQTTLens*<br>"esp/humid" → data kelembaban ditampilkan di *MQTTLens*<br>Data yang diterima akan ter-*update* setiap 1 menit | *MQTTLens* berhasil berinteraksi dengan kedua *client* melalui *publish/ subscribe* Topik<br><br>Interaksi dapat berjalan normal pada jaringan wifi dan GSM. | |
| 4.b | Kedua *client* diaktifkan kemudian unit sensor DHT11 (ESP32) didekatkan ke pemanas sampai suhu berada di atas 29oC. Kemudian dijauhkan kembali hingga suhu berada dibawah 28.5oC (algoritma Gambar 5b.) | Saat pembacaan suhu di *MQTTLens* dibawah 28.5$^{o}$C Switch Relay1 mati dan saat pembacaan suhu diatas 29$^{o}$C (DHT11 dekat pemanas) *Switch* Relay1 hidup.<br>Saat suhu kembali dibawah 28.5$^{o}$C *Switch* mati kembali. | Perubahan suhu di *Client2* mempengaruhi Relay1 *Client1* | Interaksi tetap berhasil walau kedua *Client* ditempatkan pada koneksi internet yang berbeda (wifi & gsm) |

### 3.2. Pembahasan

Fungsi *reconnect* sangat penting dan harus tersedia pada setiap *client* MQTT untuk mengantisipasi jaringan *wifi* yang sering terputus. *Client* tidak perlu di *reset* ulang agar dapat segera kembali bekerja, seting *delay* 5 detik sebelum kembali *reconnect* dapat dirubah sesuai keperluan. Fungsi *deepsleep* pada program *Client2* digunakan untuk menghemat konsumsi daya listrik apalagi jika waktu *publish* topik diset lebih lama lagi misalnya tiap 5 menit untuk sistem kembali *restart*, maka konsumsi daya akan jauh lebih hemat.

ESP8266 memiliki harga yang lebih ekonomis dari ESP32, keduanya memiliki *CPU* yang berbeda kelas, ESP3266 menggunakan *Single Core 32-bit 80Hz* (hemat daya) sedangkan ESP32 dengan prosessor *Dual Core 32-bit 160Hz*. Harga yang relatif murah berkisar Rp55.000,- s.d Rp75.000,- menjadikan ESP8266 adalah pilihan yang baik untuk sistem *IoT*. Sebenarnya rancangan kedua *client* dapat dibuat menggunakan ESP8266, akan tetapi karena penelitian ini dimaksudkan untuk menunjukkan sistem dengan model *client* yang berbeda maka digunakan ESP32 yang harganya dua kali lipat lebih mahal.

Aplikasi *MQTTLens* berperan penting untuk melakukan evaluasi sistem *MQTT* karena mampu terkoneksi ke *Broker* universal tanpa perlu konfigurasi ulang. *MQTTLens* dapat melakukan *publish* dan *subscribe* sekaligus pada topik yang sama. Data temperatur dan kelembaban udara dari *client* ESP32 akan diterima secara terus menerus tiap 1 menit sehingga data akan banyak terkumpul di rentang waktu tertentu, sayangnya *MQTTLens* tidak menyediakan tampilan data dalam bentuk grafik.

Untuk memastikan skenario *client* ESP8266 dan ESP32 dapat berinteraksi pada lokasi yang berbeda (Tabel 1. no.4a-b), maka diperlukan dua model koneksi internet yang diharapkan dapat mempresentasikan bahwa kedua *client* berada dalam jarak yang jauh satu sama lainnya. Oleh karena itu digunakan koneksi *hotspot wifi* dan *mobile-hotspot*

*smartphone* dengan jaringan berbeda. Untuk menjalankan pengujian ini tentunya harus ada penyesuaian program pada *client* karena adanya perbedaan *SSID* dan *password*.

## 4. KESIMPULAN

Dari tahapan dan skenario pengujian sistem *IoT* yang telah dilaksanakan pada penelitian ini, maka terdapat hal-hal yang dapat disimpulkan antara lain:

(1) Pada hakekatnya *client MQTT* dapat saling berinteraksi satu sama lain melalui topik yang sama pada pesan *publish dan subscribe* yang menyatu pada *MQTT Broker* di jaringan manapun asal terkoneksi ke jaringan internet yang tidak terpengaruh oleh jarak.
(2) Untuk meningkatkan keandalan koneksi ke jaringan *wifi router* maka setiap program mikrokontroler pada *client MQTT* harus dilengkapi dengan fungsi *reconnect* untuk mengurangi upaya pemeliharaan sistem akibat seringnya gangguan pada jaringan *wifi*.
(3) Karena prinsip-prinsip kerja dari model protokol *MQTT* telah berhasil ditunjukkan saat pengujian maka *prototype* sistem *IoT* yang dibuat telah bekerja sesuai harapan.

## DAFTAR PUSTAKA

Andrew K. Dennis (2015). *Raspberry Pi Home Automation with Arduino. 2nd Ed,* Packt Publishing, Birmingham B3 2PB, UK, pp 1-18.

Arduino. *Arduino IDE 1.8.6.* www.arduino.cc/en/Main/Software. Diakses: 15 Juli 2018, jam 09.30

B.S.S.Tejesh and S. Neeraja (2017). Implementation of an Efficient Smart Home System using MQTT. *International Research Journal of Engineering and Technology (IRJET).* Volume: 04 Issue: 03 Mar -2017, page 1848-1852, e-ISSN: 2395 -0056

Cloudmqtt. *Hosted message broker for the Internet of Thing*. www.cloudmqtt.com. Diakses pada tanggal 20 Juli 2018 jam 10.30

Hudan Abdur Rochman, Rakhmadhany Primananda dan Heru Nurwasito (2017). Sistem Kendali Berbasis Mikrokontroler Menggunakan Protokol MQTT pada Smarthome. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, Fakultas Ilmu Komputer, Universitas Brawijaya. Vol. 1, No. 6, Juni 2017, hlm. 445-455, e-ISSN: 2548-964X

Michael Yuan (2017). *Getting to know MQTT, Why MQTT is one of the best network protocols for the Internet of Things*. Developer Work. Published on May 12, 2017/Updated: August 07, 2017. *www.ibm.com*

Sagar Rajebhosale, Avinash Sonawane, Sunmitra Pawar, Vijay Kadam, Ashish Waghela (2017). IOT based Home Automation and Security System. *International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE)*. Vol. 6, Issue 3, March 2017, page 821-824, e-ISSN 2278-1021

Sandro. *MQTTLens Chome Aplication.* https://chrome.google.com/webstore/detail/mqttlens/hemojaaeigabkbcookmlgmdigohjobjm Diakses pada tanggal 20 Juli 2018 jam 11.00

Valerie Lampkin, Weng Tat Leong, Leonardo Olivera, Sweta Rawat, Nagesh Subrahmanyam and Rong Xiang (2012). *Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry*. 1st-ed International Technical Support Organization. IBM Redbook page 1-19