BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Penelitian sebelumnya yang berhubungan dengan *middleware* adalah penelitian yang berjudul *"Performance Evaluation of an IoT Platform"* oleh Vandikas & Tsiatsis (2014). Dalam penelitian tersebut menjelaskan tentang *middleware* yang disebut dengan IoT-Framework. Fungsi dari *middleware* ini adalah untuk menyebarkan data yang digenerate dari sebuah proses kepada para *subscriber*. IoT Framework ini menggunakan sistem RabbitMQ *publish-subscribe*. Hasil yang didapat dari penelitian ini menunjukkan bahwa IoT Framework merupakan sistem yang stabil karena tidak ada paket yang hilang saat pengujian dengan beban yang tinggi. Selain itu, penelitian ini juga menunjukkan bahwa sistem dapat menangani beban hingga 1000 *producers* / 0 *consumers* atau 1000 sensor dengan *throughput* 2173,3 (msg/sec) dalam satu node. Namun *throughput*-nya turun secara signifikan saat jumlah dari consumer meningkat. (Vandikas & Tsiatsis, 2014)

Penelitian selanjutnya adalah penelitian yang terkait dengan protocol MQTT dan CoAP yang berjudul *"Performance Evaluation of MQTT and CoAP via Common Middleware"* oleh Thangavel et al. (2014). Dalam penelitian tersebut menjelaskan tentang *Wireless Sensor Networks* (WSNs) yang biasanya terdiri dari node sensor dan gateway dengan sumberdaya yang terbatas yang menyebabkan WSN membutuhkan protokol dengan konsumsi *bandwidth* dan energi yang efisien untuk melakukan pengiriman data. Pada penelitian ini peneliti merancang dan mengimplementasikan *middleware* yang mendukung protokol MQTT dan CoAP. Berdasarkan *middleware* tersebut peneliti melakukan pengujian kinerja dari protokol MQTT dan CoAP dengan parameter *end-to-end delay* dan konsumsi bandwidth. Hasil yang diperoleh dari penelitian ini menunjukkan bahwa pesan dengan menggunakan protokol MQTT memiliki *delay* yang lebih rendah dari pesan dengan menggunakan protokol CoAP pada tingkat paket loss yang rendah dan pesan dengan menggunakan protokol MQTT memiliki *delay* lebih besar daripada pesan dengan menggunakan protokol CoAP pada tingkat paket loss yang tinggi. Pada kondisi ketika ukuran pesan lebih kecil dan loss rate kurang dari 25% CoAP menghasilkan trafik yang lebih rendah dari MQTT. (Thangavel et al., 2014)

Berdasarkan pada penelitian terdahulu penulis mengusulkan sebuah penelitian yang memiliki keterkaitan dengan penelitian sebelumnya. Penelitian yang dilakukan penulis adalah analisis kinerja IoT *middleware* pada raspberry pi zero dan raspberry pi 2. Perbandingan penelitian sebelumnya dengan penelitian ini disajikan pada Tabel 2.1.

Table 2.1 Kajian Pustaka

No	Judul	Parameter Pangujian	Perbandingan	
			Kajian Pustaka	Skripsi Peneliti
1	Performance Evaluation of an IoT Platform (Vandikas & Tsiatsis, 2014)	Throughput, Penggunaan Memori, Skalabilitas	*Middleware* yang digunakan adalah IoT-Framework	*Middleware* yang digunakan adalah Event-based *middleware*
2	Performance Evaluation of MQTT and CoAP via Common *Middleware* (Thangavel et al., 2014)	*Delay*, Packet loss, Data Transfer	*Middleware* yang digunakan adalah "common *middleware*"	*Middleware* yang digunakan adalah Event-based *middleware*
3	Analisis Performa dan Skalabilitas pada Event-based IoT *Middleware* (Rozi, 2017)	Penggunaan CPU dan memori, *Delay*, Packet loss, Skalabilitas	*Host* untuk *middleware* yang digunakan adalah raspberry pi 2	*Host* untuk *middleware* yang digunakan adalah raspberry pi zero

2.2 IoT *Middleware*

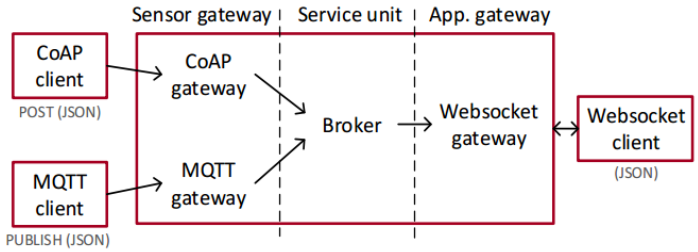
IoT *middleware* yang digunakan dalam penelitian ini berdasarkan *middleware* yang dikembangkan pada penelitian sebelumnya dengan judul "Pengembangan IoT *Middleware* Berbasis Event-Based dengan Protokol Komunikasi CoAP, MQTT, dan Websocket" oleh Anwari (2017).

2.2.1 Deskripsi Umum Sistem

Sistem yang dibangun terdiri dari satu sensor dengan protokol MQTT dan satu sensor dengan protokol CoAP yang mengirimkan data ke *middleware*. Oleh *middleware* data akan diteruskan ke web-app dan data yang dikirim dapat dimonotoring secara real-*time* melalui web-app. *Middleware* yang dibangun memiliki dua fungsi utama yaitu menyediakan gateway multi-protokol untuk pengiriman data sensor melalui protokol CoAP dan MQTT dan juga menyediakan

gateway untuk mengirim data yang diterima oleh *middleware* ke web-app melalui protokol websocket.

2.2.2 Arsitektur *Middleware*



Gambar 2.1 Arsitektur *Middleware*

Sumber: (Anwari, 2017)

Middleware yang dikembangkan menggunakan pola *publish-subscribe* dan memiliki tiga komponen utama yaitu *sensor gateway, service unit, dan application gateway*. *Sensor gateway* menyediakan *interface* untuk sensor mengirim data ke *middleware* melalui protokol CoAP atau MQTT. Komponen *service unit* menyediakan tiga fungsi utama yakni *data management, service delivery,* dan *interface definition*. Sedangkan *application gateway* menyediakan *interface* bagi aplikasi untuk terhubung ke *middleware* dan membaca data yang dikirimkan melalui protokol websocket.

1. Sensor Gateway

Sensor *gateway* berfungsi menyediakan *interface* bagi sensor untuk mengirimkan data ke *middleware*. Pada *middleware* yang dikembangkan terdapat dua jenis *gateway* yakni CoAP dan MQTT *gateway*. Pertama, CoAP *gateway* menyediakan *interface* bagi sensor untuk mengirimkan data dengan melakukan POST request ke *middleware*. Oleh Karena pada pola publish/subscribe setiap pesan yang dikirimkan harus memiliki topik sedangkan dalam CoAP tidak mengenal istilah topik, maka diperlukan sebuah metode untuk menjembatani masalah tersebut. Dalam hal ini adalah menempatkan topik sebagai bagian dari URL pada CoAP *resources* dengan semantik /r/topik. Sebagai contoh, apabila sensor hendak mengirim data untuk topik home ke *middleware* maka sensor tersebut melakukan POST *request* ke alamat coap://*host*name:port/r/home. Ketika sensor mengirimkan data, CoAP *gateway* melakukan validasi URL untuk menentukan apakah URL yang digunakan sudah sesuai dengan skema yang sudah ditentukan. Apabila valid, data tersebut akan disimpan dalam Redis oleh komponen *middleware* lainnya, namun apabila tidak maka CoAP *gateway* mengirimkan pesan *error* ke sensor. *Gateway* kedua yakni MQTT *gateway* memiliki fungsi yang sama yakni menyediakan *interface* bagi sensor untuk mengirimkan data ke *middleware*. Pada dasarnya, interaksi antara sensor dan MQTT *gateway* tidak jauh berbeda dengan COAP *gateway.* Satu-satunya

perbedaan dengan CoAP gateway adalah ketika data diterima, MQTT gateway langsung memproses data tersebut tanpa ada proses validasi. Hal ini dikarenakan sejak awal MQTT sudah menerapkan pola publish/subscribe.

CoAP *gateway* menyediakan *interface* bagi sensor untuk mengirimkan data ke *middleware*. Sensor mengirimkan data dengan melakukan POST *request* ke *middleware*. Oleh karena itu dalam CoAP *gateway* diimplementasikan sebuah fungsi untuk meng-*handle request* tersebut. Secara umum fungsi tersebut terlihat seperti pseudo-code berikut :

```
Pseudo-code 1: handlerPost
1  DEFINE function handlerPost()
2   IF URL is not valid THEN
3    RETURN 4.05
4   END
5   SET topic = get topic from URL
6   SET payload = get payload
7   DO findOrCreate(topic, payload)
8    RETURN 2.01
9  END
```

Kode Program 2.1 *Pseudo-code handlerPost*

Ketika *request* diterima, CoAP *gateway* melakukan validasi URL yang digunakan, apabila tidak valid, proses dihentikan dan kode 4.05 (*no permission*) dikirimkan ke sensor. Sebaliknya, apabila valid proses berikutnya yakni memanggil fungsi findOrCreate untuk menyimpan data yang dikirimkan ke Redis. Terakhir yakni mengirim kode 2.01 (*created*) sebagai tanda bahwa data sudah berhasil tersimpan.

Berbeda dengan CoAP *gateway*, MQTT *gateway* menggunakan paradigma *event-driven*, artinya sebuah fungsi akan dijalankan ketika ada *event* tertentu. Misalkan ketika sensor mengirimkan data, *event publish* terjadi dan fungsi pada *event* tersebut akan dijalankan. Event yang ada pada MQTT *gateway* yakni *connect*, *publish*, *disconnect*, *error* dan *close*. Event pertama yang terjadi ketika sebuah sensor terhubung dengan *middleware* yakni connect. Dalam *event* ini *middleware* mencatat ID dari sensor yang terhubung kemudian mengirimkan pesan connack untuk memberi tahu sensor bahwa koneksi sudah berhasil terbentuk.

```
Pseudo-code 2: event connect
1  ON connect DO function(packet)
2   SET id = get client id from packet
3   RETURN connack message
4  END
```

Kode Program 2.2 *Pseudo-code event connect*

Event berikutnya yakni *publish*, bisa dikatakan bahwa *event* ini merupakan yang paling utama dari MQTT *gateway*. Event ini terjadi ketiak sensor mengirimkan data ke *middleware*. Fungsi pada *event* ini sangat sederhana menyimpan data yang dikirimkan ke Redis menggunakan fungsi findOrCreate.

```
Pseudo-code 3: event publish
```

```
1  ON publish DO function(packet)
2   SET topic = get topic from packet
3   SET payload = get payload from packet
4   RETURN findOrCreate(topic, payload)
5  END
```

Kode Program 2.3 *Pseudo-code event Publish*

Event disconnect dan close terjadi ketika koneksi *client* dan *gateway* terputus baik karena *client disconnect* dengan sengaja ataupun karena hilangnya konektifitas. Sedangkan event error terjadi ketika terdapat kesalahan pada saat *client* mencoba terhubung dengan *gateway* ataupun ketika mem-*publish* data.

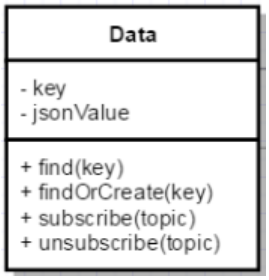
```
Pseudo-code 4: event disconnect
1  ON disconnect DO function()
2   DO end stream
3  END
4  ON error DO function(error)60
5   DO end stream
6  END
7  ON close DO function(error)
8   DO end stream
9  END
```

Kode Program 2.4 *Pseudo-code event disconnect*

2. Service Unit

Komponen ini bekerja bersama Redis untuk menjadi broker yang menjembatani komunikasi antara sensor dan *application gateway*. Service unit menyediakan *interface* bagi sensor dan *application gateway* untuk *publish* dan *subscribe* serta berinteraksi dengan Redis. Ketika *sensor gateway* mengirimkan data untuk sebuah topik, *service unit* akan menyimpan data tersebut di Redis sekaligus mem-*publish* data tersebut. Di sisi lain, setelah *application gateway subscribe* sebuah topik, apabila pada Redis terdapat data untuk topik yang diminta, *service unit* akan mengirimkan data tersebut ke aplikasi. Hal ini disebut *retain message* pada *middleware*, artinya meskipun pada saat *subscribe*, sensor sudah tak terhubung dengan *middleware*, aplikasi tersebut akan menerima data terbaru yang sudah dikirimkan sensor. Selanjutnya setiap kali sensor *gateway* mengirimkan data, *service unit* akan mengirimkan data tersebut ke aplikasi yang sudah *subscribe* untuk topik yan sama secara *real-time*.

Service layer mempunyai 3 fungsi utama yakni menyediakan *data management*, *service delivery,* dan *interface definition*. Fungsi pertama yakni *data management*, untuk hal ini Service layer mempunyai sebuah abstraksi tipe data yang akan disimpan dengan beberapa *accessors* dan *mutator method*. Struktur dari tipe data ini dapat dilihat pada diagram berikut :



Gambar 2.2 Tipe Data Service Unit

Sumber: (Anwari, 2017)

Tipe data di atas mempunyai konstruktor dengan dua parameter yakni key dan value. Dua nilai inilah yang nantinya disimpan di Redis sebagai pasangan key:value. Akan tetapi sebelumnya kedua nilai tersebut mengalami mutasi yakni key berubah menjadi topic:key dan value diformat menjadi JSON. *Service delivery* dan *interface definition* di implementasikan dengan membuat beberapa fungsi seperti Save, Find, FindOrCreate, *Publish*, *Subscribe*, dan Un*subscribe* yang nantinya berperan sebagai API bagi *gateway* untuk berinteraksi dengan Redis. *Pseudo-code* di bawah ini menjelaskan fungsi Save. Setelah nilai key dan value berhasil disimpan di Redis, selanjutnya key dan value tersebut di *publish*.

```
Pseudo-code 5: fungsi save
1   DEFINE function save(callback)
2    DO save key,value to Redis
3     DO publish key,value to Redis
4      RETURN callback
5    DO add key to a set in Redis
6   END
```

Kode Program 2.5 *Pseudo-code fungsi save*

Fungsi berikutnya yakni find. Seperti namanya, fungsi ini berfungsi untuk mencari value dari suatu topik/key pada Redis. Fungsi ini membutuhkan dua parameter yakni key dan callback. Ketentuan parameter key sama dengan topik pada MQTT yakni dapat berupa satu nama topik secara spesifik ataupun menggunakan *wildcard*. Apabila key yang digunakan adalah nama topik maka nilai dari topik tersebut langsung dikirimkan, namun apabila topik menggunakan *wildcard* data yang dikirimkan berupa array.

```
Pseudo-code 6: fungsi find
1   DEFINE function find(key, callback)
2    DEFINE function foundRecord(key)
3     SET data = get value from Redis based on key
4     IF data = null THEN
5      RETURN callback(not found)
6     ELSE
7      RETURN callback(data)
8    END
9    IF key != RegExp THEN
10    DO foundRecord(key)
11   ELSE
12    SET results = empty array
```

```
13      SET topics = get all registered topics
14      FOR EACH topic in topics
15       IF key match topic
16        SET result = foundRecord(key)
17        DO push result to results
18       END
19      END
20      RETURN results
21     END
22    END
```

Kode Program 2.6 *Pseudo-code* fungsi *find*

Fungsi findOrCreate digunakan untuk menuliskan data baru ke Redis berdasarkan satu topik tertentu. Apabila topik tersebut sudah ada, maka nilai dari topik tersebut akan ditimpa (*overwrite*), jika belum ada maka dibuat pasangan key:value baru pada Redis.

```
Pseudo-code 7: fungsi findorCreate
1     DEFINE function findOrCreate(key, value, callback)
2      SET oldValue = get value from Redis based on key
3      IF value != oldValue THEN
4       SET value as new value
5      END
6      RETURN save(callback)
7     END
```

Kode Program 2.7 *Pseudo-code* fungsi *findorCreate*

Berikutnya yakni fungsi untu *subscribe* dan un*subscribe* suatu topik pada Redis.

```
Pseudo-code 8: fungsi subscribe
1     DEFINE function subscribe(topic, callback)
2      DO subscribe to Redis with topic as key
3      RETURN callback
4     END
5     DEFINE function unsubscribe(topic, callback)
6      DO unsubscribe to Redis with topic as key
7      RETURN callback
8     END
```

Kode Program 2.8 *Pseudo-code* fungsi *subscribe*

3. Application Gateway

Application gateway menyediakan *interface* bagi aplikasi untuk membaca data dari *middleware* melalui protokol Websocket. *Interface* yang disediakan yakni *interface* untuk *connect* ke *middleware* dan *interface* untuk *subscribe* suatu topik. Ketika sebuah aplikasi hendak meminta data, pertama aplikasi tersebut harus terhubung dengan *middleware* melalui *application gateway* menggunakan *interface* connect. Selanjutnya, ketika aplikasi *subscribe* ke sebuah topik, *application gateway* akan melakukan dua hal yakni *subscribe* dan mencari data berdasarkan topik yang diminta. Apabila ditemukan data pada Redis untuk topik yang diminta, maka data tersebut akan dikirimkan ke aplikasi, selebihnya aplikasi akan menerima data secara *real-time* setiap kali sensor mengirim data untuk topik yang sama. Data tersebut berikutnya dapat di proses lebih lanjut oleh aplikasi seperti disimpan dalam basis data.

Appication gateway juga menggunakan paradigma *event-driven*, sama seperti MQTT *gateway*. *Event* yang ada pada *application gateway* yakni connection, *subscribe* dan disconnect. *Event* connection terjadi ketika terdapat aplikasi yang terhubung dengan *middleware* melalui Websocket. *Event* berikutnya yakni *subscribe*, *event* ini menjadi *interface* bagi aplikasi untuk membaca data dari sensor. Ketika event ini terjadi pertama *application gateway* melakukan *subscribe* ke Redis untuk topik yang diminta oleh aplikasi dan kedua mencari nilai dari topik tersebut dengan fungsi find. Jika ditemukan, data tersebut akan dikirimkan ke aplikasi. Hal ini sangat berguna karena setiap kali aplikasi melakukan *subscribe* ia akan menerima data saat itu juga meskipun sensor belum mengirimkan data untuk topik yang diminta. Apabila data untuk topik tersebut tidak ada, maka *application gateway* akan menunggu hingga sensor mengirimkan data kemudian meneruskannya ke aplikasi dengan emit.

```
Pseudo-code 9: event subscribe
1   ON subsribe DO function(topic)
2    DEFINE function subscription(data)
3     IF data is buffer THEN
4       DO convert to string
5     END
6     RETURN emit data to /r/ + topic
7    END
8    DO subscribe(topic, subscription)
9    RETURN find(topic)
10  END
```

Kode Program 2.9 *Pseudo-code event subscribe*

Event selanjutnya yakni disconnect yang terjadi ketika koneksi antara aplikasi dan *middleware* terputus. Fungsi pada *event* ini akan melakukan un*subscribe* topik dari aplikasi yang terputus.

```
Pseudo-code 10: event disconnect
1   ON disconnect DO function()
2    DO unsubsccribe(topic)
3   END
```

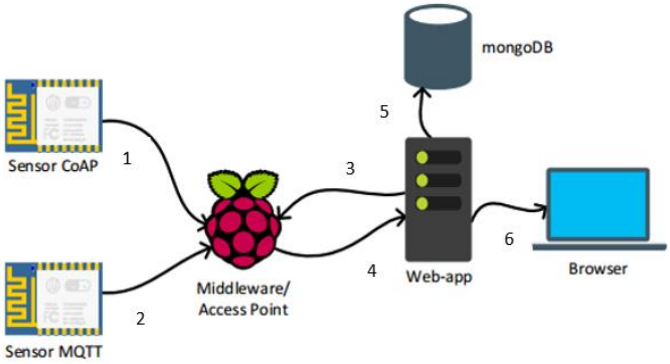
Kode Program 2.10 *Pseudo-code event disconnect*

2.2.3 Alur Komunikasi Sistem

Terdapat tiga komponen utama dalam sistem yakni raspberry pi sebagai *middleware*, sensor suhu dan kelembapan sebagai *publisher*, dan laptop sebagai *subscriber* dengan menjalankan aplikasi web. Ada dua interaksi komunikasi sistem yaitu pengiriman data dari sensor ke *middleware* dan pengiriman data dari *middleware* ke web-app.

Pertama sensor harus terhubung dengan *middleware* supaya dapat mengirimkan data suhu dan kelembapan. Interval pengiriman data dilakukan setiap 30 detik sekali. Mekanisme pengiriman data sensor MQTT dengan melakukan *publish* ke *middleware* dengan topik `home/barrack`, sedangkan untuk data sensor CoAP dengan melakukan POST *request* ke *middleware* dengan topik `home/kitchen`.

Setiap kali sensor mengirimkan data, *middleware* akan mengirimkan data tersebut secara *realtime* ke web-app menggunakan protokol websocket. Sebelum web-app dapat menerima data dari *middleware*, web-app harus terhubung terlebuh dahulu dengan *middleware* dan melakukan *subscribe* pada topik yang dikirimkan oleh sensor. Setelah data diterima oleh web-app, data tersebut akan disimpan dalam MongoDB.



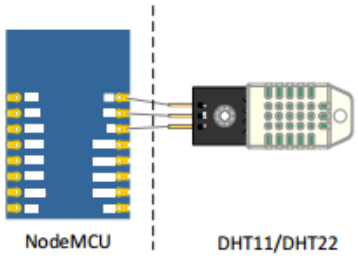
Gambar 2.3 Alur Komunikasi Sistem

Sumber: (Anwari, 2017)

Keterangan pada Gambar 2.3 diatas adalah:

1. Sensor CoAP mengirimkan data suhu dan kelembapan ke *middleware* dengan topik home/kitchen setiap 30 detik sekali.
2. Sensor MQTT mengirimkan data suhu dan kelembapan ke *middleware* dengan topik home/barrack setiap 30 detik sekali.
3. Web-app *subscribe* topik home/kitchen dan home/barrack ke *middleware*.
4. Web-app menerima data suhu dan kelembapan untuk topik home/kitchen dan home/barrack.
5. Web-app menyimpan data kedalam database mongoDB.
6. Web-app diakses menggunakan *browser* pada laptop.

2.2.4 Node Sensor



Gambar 2.4 Node Sensor

Sumber: (Anwari, 2017)

Node sensor dibangun menggunakan dua jenis perangkat keras yaitu nodeMCU sebagai mikrokontroler dan modul DHT11/DHT22 sebagai sensor suhu dan kelembapan seperti ditunjukkan pada Gambar 2.4. Pada nodeMCU akan ditanamkan kode program berbasis LUA sehingga dapat membaca nilai suhu dan kelembapan dari sensor DHT yang kemudian dikirim melalu protokol MQTT atau CoAP. Data yang dikirimkan oleh nodeMCU berbentuk JSON dengan skema sebagai berikut:

```
{
  protocol = string,  // protocol using
  timestamp = string, // fetched from time server
  topic = string,
  sensor = {
    tipe = string,    // keyword type is reserved.
    index = string,   // sensor index
    ip = string,      // node's ip address
    module = string   // dht11 or dht22
  },
  humidity = {
    value = number,
    unit = string     // in percentage
  },
  temperature = {
    value = number,
    unit = string     // celsius or fahrenheit
  }
}
```

Kode Program 2.11 Semantik Data

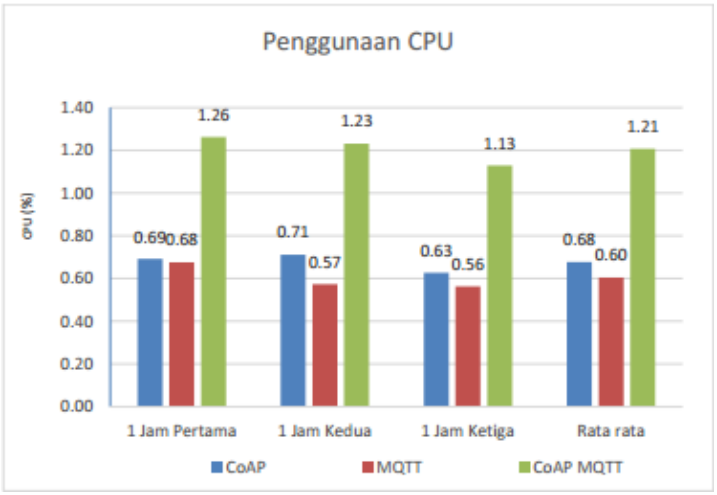
Sumber: (Anwari, 2017)

2.3 Kinerja *Middleware*

Pada penelitian sebelumnya yang berjudul "Analisis Performa dan Skalabilitas pada Event-based IoT *Middleware*" oleh Rozi (2017) telah dilakukan pengujian terhadap *middleware* dengan menggunakan raspberry pi 2 sebagai *host*-nya. Parameter pertama pengujian yang digunakan dalam penelitian ini adalah parameter performansi yang meliputi penggunaan CPU dan memori, *delay* dari sensor ke *middleware*, *delay* dari *middleware* ke web-app, dan *end-to-end delay*. Parameter kedua yang digunakan dalam penelitian ini adalah parameter skalabilitas yang menganalisis bagaimana kinerja dari *middleware* ketika jumlah *publisher* atau *subscriber* meningkat.

2.3.1 Performansi

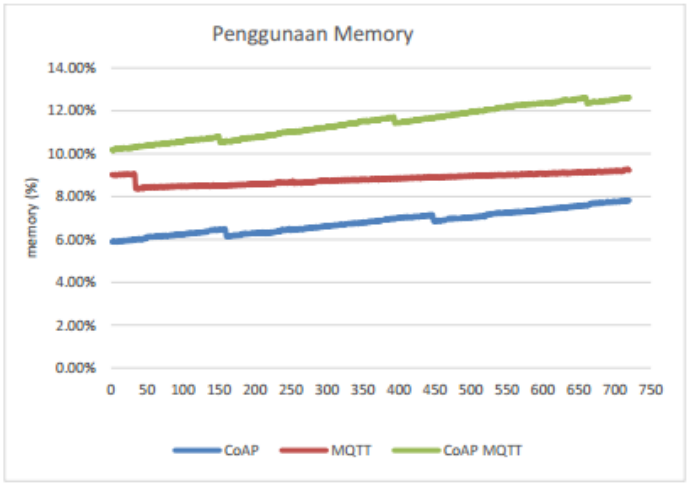
Dalam pengujian performansi yang dilakukan dibagi menjadi pengujian penggunaan CPU dan memori, pengujian *middleware* terhadap *delay*, *end-to-end delay*, dan paket loss.



Gambar 2.5 Grafik Penggunaan CPU

Sumber: (Rozi, 2017)

Gambar 2.5 menunjukkan hasil dari penggunaan CPU selama 3 jam. Rata-rata penggunaan CPU ketika menggunakan MQTT saja sebesar 0,60%, ketika menggunakan CoAP saja sebesar 0,68%, dan ketika menggunakan CoAP dan MQTT bersamaan sebesar 1,21%. Dari hasil tersebut dapat disimpulkan bahwa penggunaan CPU oleh *middleware* tergolong kecil dimana hanya menggunakan rata-rata 1,21% dari CPU.



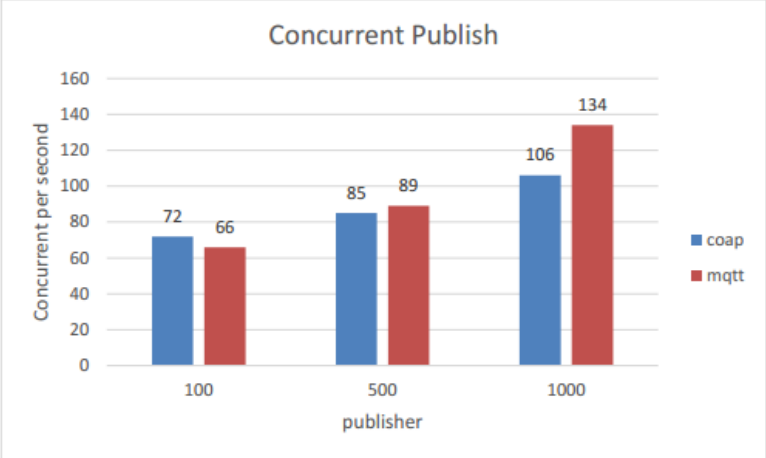
Gambar 2.6 Grafik Penggunaan Memori

Sumber: (Rozi, 2017)

Pada Gambar 2.6 menunjukkan hasil dari penggunaan memori selama 3 jam. Penggunaan memori ketika menggunakan CoAP saja sekitar 5,87%-7,83%, ketika menggunakan MQTT saja sekitar 8,35%-9,30%, dan ketika menggunakan MQTT dan CoAP secara bersamaan penggunaan memori sekitar 10,15%-12,63%.

2.3.2 Skalabilitas

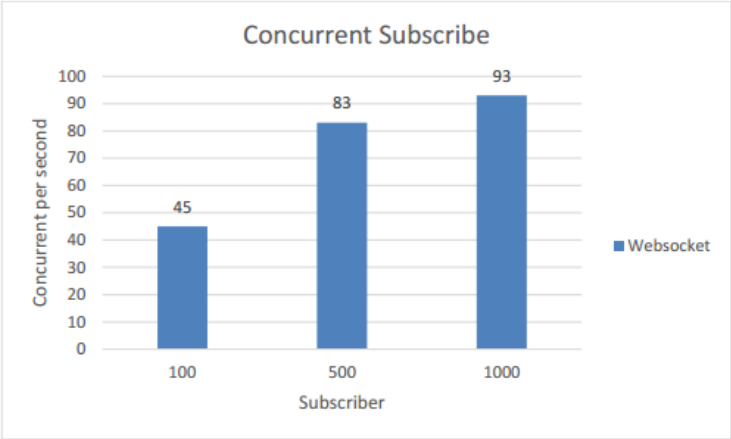
Pengujian skalabilitas dilakukan untuk mengetahui bagaimana kinerja dari *middleware* dalam menangani peningkatan jumlah *publisher*/*subscriber*.



Gambar 2.7 Grafik *Concurrent Publish*

Sumber: (Rozi, 2017)

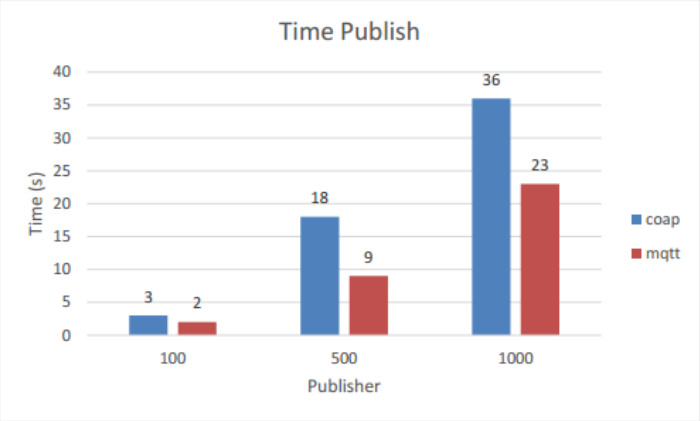
Pada Gambar 2.7 menunjukkan banyaknya *publish* yang dapat ditangani oleh *middleware* per detiknya. Baik protokol MQTT maupun CoAP mengalami peningkatan *concurrent publish* seiring dengan meningkatnya jumlah *publisher*. Dari sisi klien, *publisher* yang menggunakan protokol MQTT lebih baik daripada *publisher* dengan protokol CoAP.



Gambar 2.8 Grafik *Concurrent Subscribe*

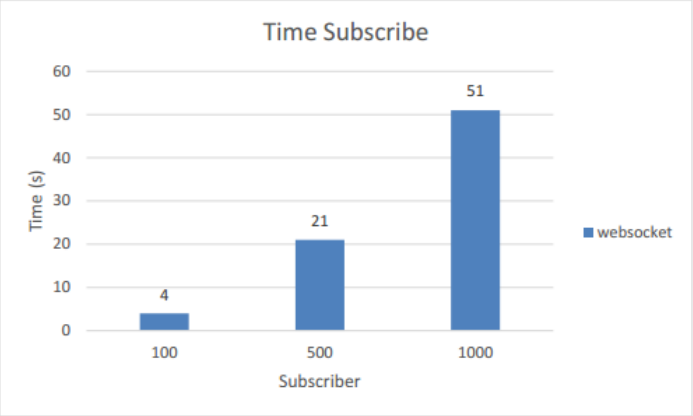
Sumber: (Rozi, 2017)

Gambar 2.8 menunjukkan banyaknya *subscriber* yang dapat ditangani oleh *middleware* per detiknya. Jumlah dari *concurrent subscriber* yang dapat ditangani meningkat seiring dengan meningkatnya jumlah *subscriber*. Hal ini dikarenakan mekanisme kerja dari websocket dalam hal ini adalah dengan membuat koneksi sebanyak jumlah *subscriber*. (Rozi, 2017)



Gambar 2.9 Grafik *Time Publish*

Sumber: (Rozi, 2017)



Gambar 2.10 Grafik *Time Subscribe*

Sumber: (Rozi, 2017)

Pada Gambar 2.9 dan Gambar 2.10 menunjukkan waktu yang dibutuhkan oleh *middleware* untuk menyelesaikan *query publish/subscribe*. Waktu yang dibutuhkan untuk menyelesaikan *query publish/subscribe* berbanding lurus dengan banyaknya jumlah *publisher*/*subscriber*. Semakin banyak jumlah *publisher*/*subscriber* semakin banyak pula waktu yang dibutuhkan untuk menyelesaikan *query publish/subscribe*.

2.3.3 Metode Pengujian

Pada penelitian ini pengambilan data untuk penggunaan CPU dan memori adalah dengan menjalankan program yang dapat mencatat penggunaan CPU dan memori berdasarkan PID (*process ID*) dari *middleware* setiap 15 detik sekali (Kode Program 2.12).

```
var usage = require('usage');

var pid = 24571

setInterval(function() {

        var options = { keepHistory:true);
   usage.lookup(pid, options, function(err, stat) {
        console.log(err, stat);
        console.log(new Date().toISOString());

   });
},15000);
```

Kode Program 2.12 Penggunaan CPU dan memori

Sedangkan untuk pengujian skalabilitas dilakukan dengan menggunakan package async dari npm yang dapat menjalankan program secara asynchronous. Program pengiriman data (*publish*) dan program meminta data (*subscribe*) dimasukkan ke dalam async dan diatur berapa banyak program yang ingin dijalankan secara asynchronous (Kode Program 2.13).

```
// generate 5 users
async.times(5, function(n, next) {
    //program untuk publih/subscribe
    Publisher() {
        next();
    });
}, function(err) {

});
```

Kode Program 2.13 Skalabilitas

Berdasarkan metode tersebut, maka pada penelitian ini dilakukan cara yang sama untuk mengambil data sehingga didapatkan data yang berimbang dan dapat dijadikan sebagai perbandingan.