Analisis Performasi Metode Load Balancing pada Broker Protokol MQTT Menggunakan Algoritma Least Connection

Sepriano[1], Vera Suryani[2], Erwid Musthofa Jadied[3]

[1,2,3]Fakultas Informatika, Universitas Telkom, Bandung
[1]sepriano@students.telkomuniversity.ac.id, [2] verasuryani@staff.telkomuniversity.ac.id,
[3]jadied@staff.telkomuniversity.ac.id

Abstrak

IoT di dukung oleh banyak protokol salah satunya adalah *Massage Queueing Telemetry Transport* (MQTT). MQTT adalah protokol komunikasi yang membutuhkan *resource* dan *bandwidth* yang kecil. Protokol MQTT menggunakan *broker* yang bertugas untuk menghubungkan *publisher* dan *subscriber*. Ketika *broker* mengalami kegagalan, *publisher* dan *subscriber* tidak dapat melakukan proses komunikasi dan harus menunggu untuk *broker* di perbaiki. Permasalahan ini dapat diminimalisir dengan mengimplementasikan metode *load balancing. Load balancing* merupakan teknik mendistribusikan beban *traffic* pada dua atau lebih server agar terjadi pemerataan *traffic* dan menghidari *overload* terhadap kemungkinan yang terjadi di server. *Least connection* merupakan salah satu algoritma dari *load balancing* dimana algoritma ini bekerja berdasarkan koneksi yang dilayani oleh server. Algoritma *least connection* bekerja perhitungan yang lebih kompleks dengan membandingkan jumlah koneksi pada setiap server. Hasil pengujian yang dilakukan algoritma *least connection* mendapatkan nilai parameter *throughput* yang tinggi dan *error rate* yang lebih rendah.

Kata kunci : MQTT, Load Balancing, Least connection.

Abstract

IoT is supported by many protocols, one of which is Massage Queuing Telemetry Transport (MQTT). MQTT is a communication protocol that requires small resources and bandwidth. The MQTT protocol uses a broker whose job it is to connect publishers and subscribers. When the broker experiences a failure, the publisher and subscriber cannot communicate and have to wait for the broker to be repaired. This problem can be minimized by implementing load balancing methods. Load balancing is a technique of distributing traffic loads on two or more servers so that traffic is evenly distributed and avoid overloads that may occur on the server. Least connection is one of the load balancing algorithms in which this algorithm works based on the connection served by the server. The least connection algorithm works a more complex calculation by comparing the number of connections on each server. The results of the test using the least connection algorithm get a high throughput parameter value and a lower error rate.

Keywords : MQTT, Load Balancing, Least connection.

1. Pendahuluan

1.1 Latar Belakang

Internet of Things (*IoT*) merupakan salah satu teknologi yang berkembang pesat saat ini dengan konsep menghubungkan "*Things*" atau benda-benda nyata di dunia ini dengan internet sebagai media saling berbagi informasi[6]. Pada implementasinya, *IoT* di dukung oleh banyak protokol salah satunya adalah *massage queuing telemetry transport*. Protokol *MQTT* merupakan protokol yang *open, simple*, ringan, dan protokol yang mudah diimplementasikan untuk *messaging*. Sehingga karakteristik ini membuat *MQTT* ideal untuk *client* dengan lingkungan dimana *bandwith networ*k dan *device* dengan kemampuan memori dan proses yang rendah (Dhall & Solanki, 2017). Protokol *MQTT* berjalan diatas *TCP/IP*. Protokol ini menggunakan medel *publish-subscribe* untuk melakukan komunikasi antara *device* yang ada. Model *Publish-Subscribe* didesain agar mudah untuk diimplementasikan dan bersifat open (Dhall &Solanki, 2017). Protokol *MQTT* menggunakan *MQTT server* sebagai perantara/jembatan antara *publisher* dan *subscriber* yang disebut *broker*.

*Mosquitt*o merupakan aplikasi *open sourc*e yang dapat digunakan sebagai *broker* pada protokol *MQTT*. *Mosquitto* mendukung *bridge*, yaitu mekanisme agar setiap *broker* dapat saling terhubung dan bertukar pesan. *Mosquitto* tidak memiliki mekanisme untuk mengatasi kegagalan yang disebabkan oleh *overload* pada *CPU* dan kerusakan perangkat keras[4]. Sehingga ketika terjadi kegagalan maka *client* yang terhubung dengan *server* kehilangan akses terhadap data yang dimiliki oleh *server* tersebut[4]. Sebuah jaringan yang memiliki distribusi *load* yang merata akan membantu optimasi terhadap *resource* yang tersedia untuk dapat

memaksimalkan *throughput*, meminimalisasi *response time*, dan mencegah terjadinya *overload* pada jaringan (Zha et al. 2010). Untuk mendistribusikan *load* pada jaringan tersebut dibutuhkan *load balancer* agar mendistribusikan *traffic* yang ada pada jaringan tersebut secara merata ke *broker* lain sesuai dengan algoritma yang diterapkan. Jadi dapat dikatakan bahwa *load balancer* memiliki fungsi utama untuk mencegah *congestion* serta memangkas *delay* yang tidak diperlukan (Boero et al. 2016). *Load balancer* memiliki memiliki mekanisme *balancing* menggunakan algoritma *least connection.* Algoritma *least connection* melakukan pembagian beban berdasarkan banyaknya koneksi yang sedang dilayani oleh sebuah server. Pada server yang memiliki kemampuan pemrosesan yang sama, algoritma penjadwalan *least connection* akan mendistribusikan beban permintaan dengan baik karena permintaan yang panjang tidak akan disalurkan ke sebuah server (Kurniawan, 2013). Sehingga dengan algorima *least connection server* dapat memberikan respon yang baik terhadap permintaan yang dilakukan oleh *user* dan juga dapat digunakan sebagai penyeimbang beban pada *server*[3].

Salah satu aplikasi *load balancer* adalah *HAProxy*. *HAProxy* merupakan aplikasi *open source* yang dapat melakukan distribusi beban kerja berdasarkan 2 jenis paket yaitu *HTTP* dan *TCP* [4]. Aplikasi *HAProxy* diinstall pada perangkat keras *raspberryPI 3B+* agar bisa bertugas sebagai *gateway* dalam perancangan sistem. Hal ini dilakukan karena tugas utama *gateway* adalah sebagai destinasi pengiriman data dari *client* untuk diteruskan kepada *broker server.*

Pada penelitian ini cara kerja yang ditawarkan adalah *publisher* mengirimkan pesan pada *gateway* dengan menggunakan *IP mikrokomputer raspberryPI 3B+* kemudian *load balancer* meneruskan pesan tersebut ke *broker* sesuai algoritma *load balancing* yang digunakan. Kemudian *broker* yang menerima pesan akan menyebarkan pesan tersebut ke *broker* lain yang terhubung menggunakan *bridge mosquitto* sehingga semua *broker server* menerima pesan yang sama. *Subscriber* melakukan *subscribe* melalui *gateway,* dimana *load balancer* akan mengarahkan pesan *subscriber* ke *broker* sesuai dengan cara kerja algoritma *least connection*.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dijabarkan, rumusan masalah dalam penelitian ini diantaranya, yaitu:

1. Bagaimana *load balancing* algoritma *least connection* dapat bekerja pada protokol MQTT?
2. Bagaimana kerja *load balancing* dapat meminimalisisr kegagalan yang terjadi pada *broker*?
3. Bagaimana implementasi dari *load balancing* dan algoritma *least connection*?

1.3 Tujuan

Tujuan dari penulisan tugas akhir ini diantaranya:

1. Implementasi metode *load balancing* pada protokol MQTT
2. Untuk mengukur performasi beban pada server MQTT
3. Mengetahui potensi performansi pada broker server menggunakan metode *load balancing* algoritma *least connection*

1.4 Batasan Masalah

Topik tugas akhir yang diangkat mengenai perancangan sistem menggunakan metode *load balancing* pada *broker* protokol *MQTT* algoritma *least connection*. Perancangan ini dibuat agar sistem dapat melakukan pembagian berdasarkan banyaknya koneksi yang sedang dilayani oleh *server*. *Server* yang paling sedikit melayani koneksi *client* maka server tersebut yang akan mengambil *request* berikutnya dari *client*. Fokus penelitian ini adalah menanggulangi permasalahan terhentinya komunikasi antara *publisher* dan *subscriber* dengan *broker* sehingga beban *traffic* dapat didistribusikan oleh sistem pada *broker* yang lain sehingga komunikasi yang mengalami permasalahan dapat di atasi. Selain itu perancangan sistem ini dibuat agar mencegah terjadinya *overload* pada satu *broker server*. Batasan masalah dalam penelitian ini adalah sebagai berikut:

1. Rancangan sistem yang dibuat adalah simulasi.
2. Menggunakan *mikrokomputer raspberryPi 3B+* sebagai *gateway*.
3. Metode yang digunakan adalah *load balancing* dengan *software HAProxy* dengan algoritma *least connection*.

1.5 Organisasi Tulisan

Bagian-bagian pada tugas akhir ini terdiri dari: Bab 1 menjelaskan mengenai latar belakang, rumusan masalah, tujuan, batasan masalah dan organisasi tulisan. Bab 2 membahas teori-teori yang berkaitan dengan penelitian. Bab 3 mengenai sistem yang dibangun. Bab 4 membahas mengenai penelitian dan evaluasi. Dan pada bab 5 adalah kesimpulan dan saran yang bisa diambil dari penelitian.

2. Studi Terkait

2.1 Penelitian Sebelumnya

Protokol mqtt adalah protokol yang secara spesifik didesain untuk device yang memiliki keterbatasan *resources* dengan *high-latency* dan *low-bandwitdh* (Happ & Wolisz, 2016).

Berdasarkan penelitian yang telah dilakukan berjudul "Analisis performa *load balancing* pada *broker* MQTT menggunakan algoritma *round robin*" di dapat pernyataan bahwa sebuah jaringan yang memiliki distribusi *load* yang merata akan membantu optimasi terhadap *resource* yang tersedia untuk dapat memaksimalkan t*hroughput*, meminimalisasi *resource time*, dan mencegah terjadinya *overload* pada jaringan (Zha et al. 2010)**.** Pada penelitian ini penulis melakukan analisis *load* dengan *load balancing* dan *client* yang digunakan adalah *script python*. *Load balancing* dalam mendistribusikan *load* bergantung dengan jumlah *broker*[4].

Load balancing adalah sebuah teknik mendistribusikan beban *traffic* pada dua jalur atau lebih, sehingga didapatkan sambungan yang seimbang, *traffic* yang lebih optimal, *throughput* data maksimal, *delay* minimal, serta tidak terjadi *overload*. *Load balancing* dapat diimplementasikan pada perusahaan yang memiliki minimal dua sambungan internet. Penelitian terkait menyatakan bahwa beban jaringan tidak menjadi lebih ringan dengan adanya *load balancing*, akan tetapi *load balancing* bertugas sebagai pengatur alokasi beban jaringan(Saputra, 2013). Salah satu aplikasi *load balancing* adalah *HAProxy*. *HAProxy* dapat melakukan distribusi beban kerja berdasarkan 2 jenis paket, yaitu paket *HTTP* dan paket *TCP[4]*.

Dengan adanya *Load balancing*, realibilitas dan ketersediaan suatu *resource* bisa lebih terjamin, karena ketika terjadi kegagalan pada salah satu broker, maka *load balancing* bisa mengarahkan *traffic* ke broker lain yang masih bisa bekerja.

Gateway merupakan bagian penting, karena *gateway* merupakan penghubung antara *client* dengan *broker server*. Oleh sebab itu tingkat ketersediaan *gateway* haruslah tinggi untuk tetap dapat meneruskan data ke *broker* ketika terjadi gangguan[1].

Pada prinsip dasarnya, suatu komunikasi data merupakan proses mengirimkan data dari satu komputer ke komputer yang lain untuk terselenggaranya proses pengiriman paket data tersebut. Untuk komunikasi antar protokol IoT dapat menggunakan arsitektur *publish/subscribe* atau *request/reply*[5].

Untuk mendistribusikan *load* yang ada, maka *load balancing* diperlukan untuk mendistribusikan beban sehingga beban *broker* bisa lebih merata. *Load balancing* adalah *device* yang memiliki kemampuan untuk mendistribusikan *traffic* yang ada dan mengarahkan *request* ke server yang ada sesuai dengan algoritma yang diterapkan. Jadi dapat dikatakan bahwa *load balancing* memiliki fungsi utama untuk mencegah *congestion* serta memangkas *delay* yang tidak diperlukan (Boero et al. 2016).

2.2 MQTT (Message Queueing Telemetry Transport)

Pada model komunikasi protokol *MQTT*, *broker* memegang peranan penting pada keberhasilan proses komunikasi yang terjadi. Hal itu disebabkan karena komunikasi antara *publisher* dan *subscriber* terjadi secara asinkron yang artinya komunikasi yang terjadi harus melalui perantara *broker* (Hayun & Wibisono, 2017). Akan tetapi, *broker* yang menjadi jembatan komunikasi memiliki kemungkinan tidak dapat tersedia. Hal itu dapat disebabkan karena *broker* memiliki permasalahan pada jaringan atau perangkat keras yang digunakan. *Publisher* dan *subscriber* akan kehilangan *service* ketika *broker* tidak tersedia yang menyebabkan *publisher* tidak dapat melakukan *publish* informasi dan *subscriber* tidak dapat melakukan *subscribe* suatu *topic*.

2.3 Publisher dan Subscriber

Publisher dan *subscriber* merupakan komponen penting pada protokol MQTT. *Client* yang terhubung menggunakan *topic* dan memanfaatkan *broker* sebagai jembatan. Tugas dari *publisher* adalah mengirimkan data yang didapatkan dari *client* dan untuk *dipublish* ke *gateway*. Sedangkan *subscriber* adalah menarima data yang dikirimkan *publisher* untuk ditampilkan pada mqtt *client*. Dalam penelitian ini digunakan *MQTTbox* sebagai *client* yang melakukan *publish subscriber* dan terhubung ke *gateway*.

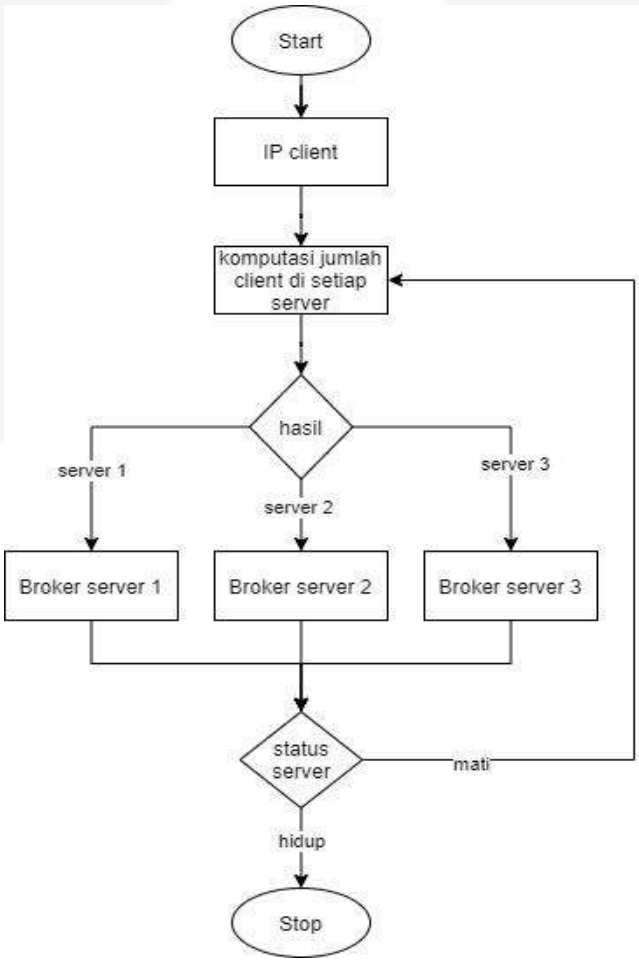
2.4 Load Balancing

Load balancing adalah suatu teknik yang digunakan untuk memisahkan antara dua atau banyak *link*. Dengan mempunyai banyak *link* maka optimalisasi untilitas sumber daya, *throughput* atau *response time* akan semakin baik karena mempunyai lebih dari satu *link* yang bisa saling mem-*backup* pada saat *network down* dan menjadi cepat pada saat *network* normal. Jika memerlukan realibilitas tinggi yang memerlukan 100% koneksi uptime dan yang menginginkan koneksi *upstream* yang berbeda dan dibuat saling mem-*backup* (Lukitasari dan Oklilas, 2010).

Load balancing adalah teknik untuk mendistribusikan beban *traffic* pada dua atau lebih jalur koneksi secara seimbang, agar *traffic* dapat berjalan optimal, memaksimalkan *throughput*, memperkecil waktu tanggap dan menghindari *overload* pada saat salah satu jalur koneksi (Sirajuddin, 2012).

2.5 Least Connection

Algoritma *least connection* melakukan pembagian beban berdasarkan banyaknya koneksi yang sedang dilayani oleh sebuah server. Server dengan koneksi yang paling sedikit akan diberikan beban berikutnya, begitu pula server dengan koneksi banyak akan dialihkan bebannya ke server lain yang bebannya lebih rendah (Ellrod, 2010).



Gambar 2.5.1 diagram alir *least connection*

Pada gambar 2.5.1 menjelaskan mengenai alur kerja algoritma *least connection,* dimana diawali dengan sistem menerima *request client*, sebelum sistem mengarahkan pada salah satu *broker server* sistem akan melakukan komputasi pada setiap *broker server* untuk mencari jumlah koneksi terkecil pada semua *broker server*. Setelah itu *request client* akan diarahkan pada jumlah koneksi *broker server* paling sedikit. Setelah proses pendistribusian antara *request client* ke *broker server* selesai, terdapat kondisi dimana ketika terjadi gangguan (*server down*) terhadap salah satu *broker server* maka koneksi pada *broker server* tersebut akan dialihkan pada *broker server* lainnya.

3. Sistem yang Dibangun

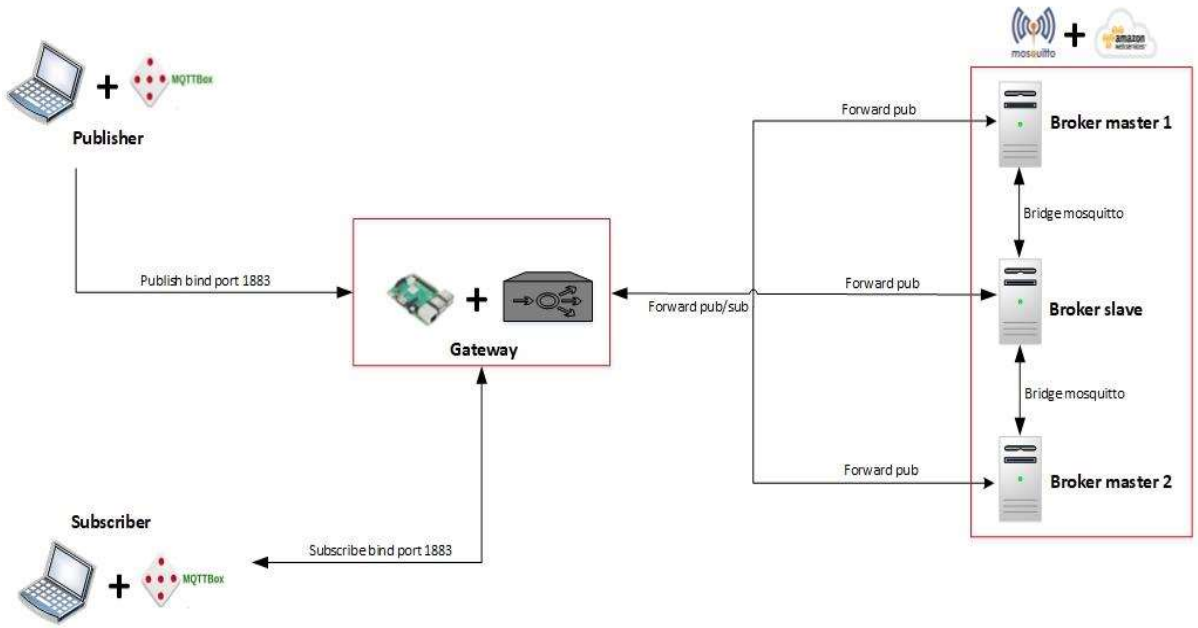
Pada pengerjaan Tugas Akhir ini, langkah awal dimulai dari identifikasi masalah yang ditemukan, kemudian dilakukan studi literatur terhadap permasalahan sehingga didapatkan teori-teori yang mendukung memecahkan masalah tesebut. Setelah itu dilakukan proses perancangan sekaligus analisis kebutuhan kompenen-komponen yang dibutuhkan dalam perancangan sistem. Kemudian diimplementasikan sesuai dengan rancangan yang sudah dibuat, selanjutnya dilakukan hasil pengujian dan dianalisa terhadap sistem. maka di dapat suatu kesimpulan untuk menjawab permasalahan yang ada.

Gambar 3.1 metodologi penelitian

3.1 Arsitektur Sistem

Rancangan umum sistem yang akan dibangun adalah sistem yang mampu mendistribusikan beban *traffic* saat *publisher-subscriber* melakukan komunikasi. Sistem ini menggunakan metode distribusi pesan menggunakan model *publisher-subscriber* yang berjalan menggunakan protokol komunikasi *MQTT*. *Broker* server pada protokol *MQTT* memiliki peran penting ketika melakukan komunikasi, mengingat pentingnya *broker* maka dibangun sebuah sistem yang dapat membantu kinerja *broker* server apabila terjadi kegagalan

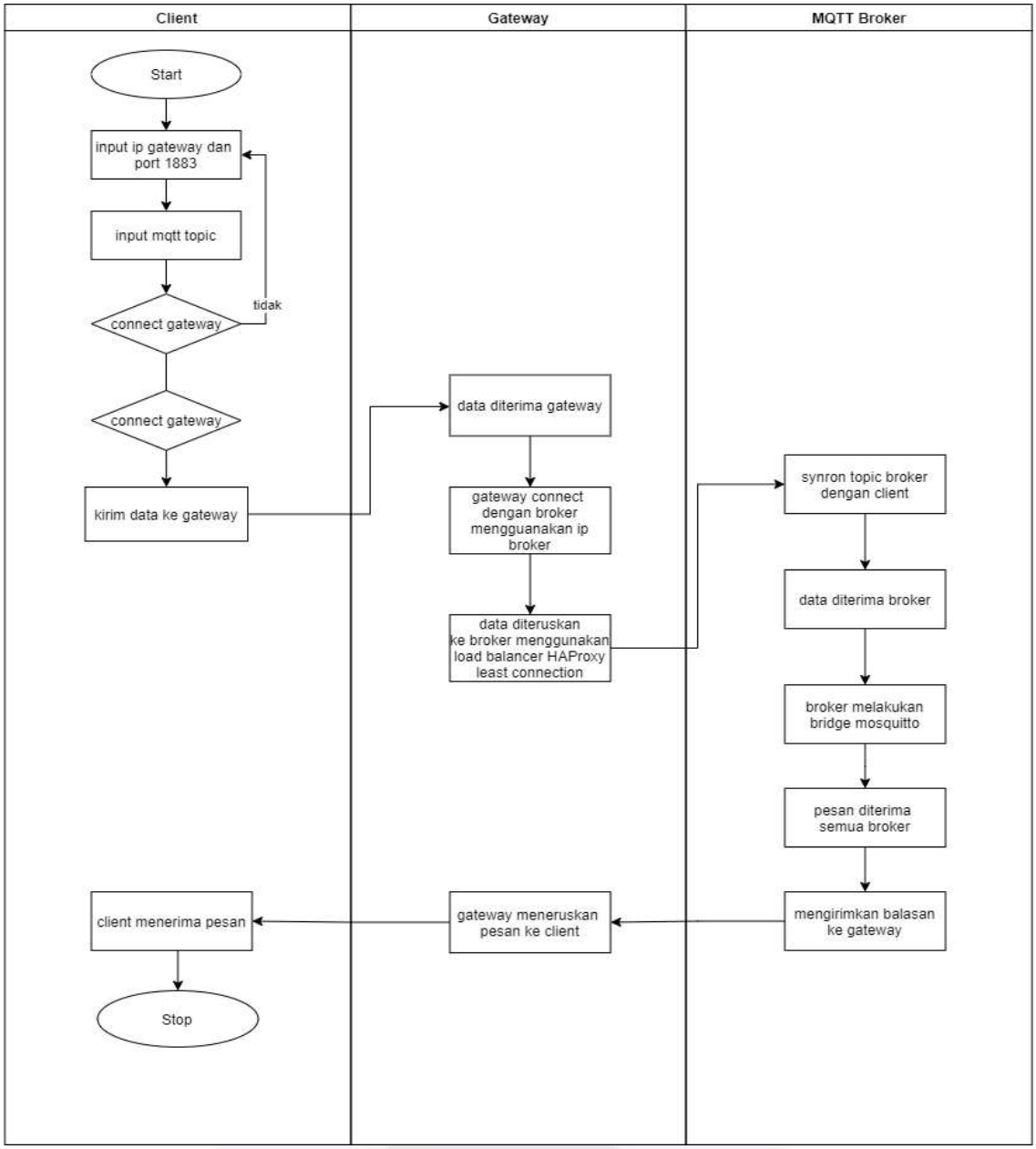
dalam pelaksanaan kerjanya. Dengan menggunakan metode *load balancing* sistem dapat mendistribusikan pesan komunikasi yang dilakukan *publisher* kepada banyak *broker server*, apabila terjadi kegagalan terhadap satu *broker server* maka *broker server* lain dapat menggantikan untuk mendistribusikan pesan komunikasi tersebut agar bisa sampai pada tujuannya. Selain itu, waktu tanggap untuk pendistribusi pesan terhitung singkat karena beban *traffic* dibagi ke banyak *broker server*.



Gambar 3.1.1 Perancangan sistem secara umum

Pada gambar 3.1.1 merupakan alur yang perlu diperhatikan saat perancangan sistem. Antara *client* dan *gateway* berada pada jaringan lokal dengan menggunaka *bind* port 1883. Kemudian data yang diterima oleh *gateway* akan diteruskan pada *broker server* dengan menggunakan *software open source HAProxy* yang dilakukan konfigurasi menggunakan algoritma *least connection*. Penjelasan lebih rinci adalah sebagai berikut:

1. *Client* dan *gateway* haruslah berada pada satu jaringan lokal yang memanfaatkan modul komunikasi nirkabel *WiFi*.
2. *Broker server* MQTT yang berjalan pada *virtual server* akan berjalan secara bersamaan pada *ip public* yang berbeda.
3. *Broker server* tersebut akan melakukan sinkronisasi *topic* memanfaatkan *bridge mosquitto* agar tiap *broker server* mendapatkan pesan yang dikirim oleh *gateway* pada *topic* yang telah ditentukan.
4. *Gateway* memanfaatkan mikrokomputer *raspberryPI 3B+* sebagai pusat pemrosesan data yang dikirim oleh *client* dan nantinya akan diteruskan pada *broker* menggunakan *HAProxy*.
5. *Client* memanfaatkan modul *WiFi* sebagai media pengiriman data, akan melakukan *sensing* data yang selanjutnya akan diteruskan atau dikirim kepada *gateway* melalui mekanisme *publish* dengan topik yang telah ditentukan dan menggunakan alamat *ip gateway* sebagai alamat host pengiriman.
6. *Gateway* sebagai destinasi pengiriman oleh *client* memanfaatkan *load balancer HAProxy* sebagai portal pengiriman ke *broker server*.
7. *HAProxy* dan *least connection* akan bertugas melakukan *bind port MQTT* yang berada pada *port 1883* di *broker server* dan juga membagi beban *traffic* dari *client* yang mencoba melakukan *publish* ke *broker server* dengan memanfaatkan *load balancer*.
8. Ketika *topic* yang dikirimkan oleh *client* kepada *gateway* sesuai dengan *topic* yang telah disinkronisasikan maka salah satu *broker server* akan menerima pesan tersebut, kemudian melakukan sinkronisasi kepada *broker server* lainnya sehingga semua *broker server* mendapatkan pesan yang dikirim oleh *client* dan pembagian beban *traffic* yang sama sesuai dengan algoritma *least connection*.



Gambar 3.1.2 Diagram alir *client* melakukan *publish/subscribe*

Gambar 3.1.2 menggambarkan alur proses yang dilakukan *client* ketika melakukan *publish/subscribe* ke *broker server* pada perancangan sistem. Ketika *client* melakukan *request, request* tersebut tidak ditangani langsung oleh *broker server* tetapi ditangani terlebih dahulu oleh *gateway* yang diinstall *load balancer HAProxy*. Pada *load balancer HAProxy* dilakukan konfigurasi yang diterapkan pada *HAProxy* dimana konfigurasi tersebut memuat tentang algoritma yang digunakan yaitu *least connection*.

Proses pertama dimulai dengan *client* terhubung dengan *gateway raspberryPI 3B+* menggunakan IP *raspberryPI 3B+* dan *port 1883* dengan *topic* yang sama antara *client* dan *broker server*. Setalah terhubung maka *client* dapat melakukan proses *publish* data ke *broker server* melalui *gateway*. Apabila gagal terhubung maka *client* melakukan *reconnect* ulang. Selanjutnya pada *sudo nano /etc/haproxy/haproxy.cfg* dibuat konfigurasi sehingga *HAProxy* dapat melakukan tugasnya yaitu membagi *traffic* pada *broker server* bedasarkan algoritma *least connection*.

```
# Listen to all MQTT request (port 1883)
listen mqtt
        #MQTT binding to port 1883
        bind *:1883
        mode tcp
        option tcplog
        # balance mode (to choose which MQTT server to use)
        balance leastconn
        # MQTT broker master 1
        server Broker-master-1 3.88.252.106:1883 check
        # MQTT broker slave
        server Broker-slave 3.86.105.209:1883 check
        # MQTT broker master 2
        server Broker-master-2 52.91.64.120:1883 check
```

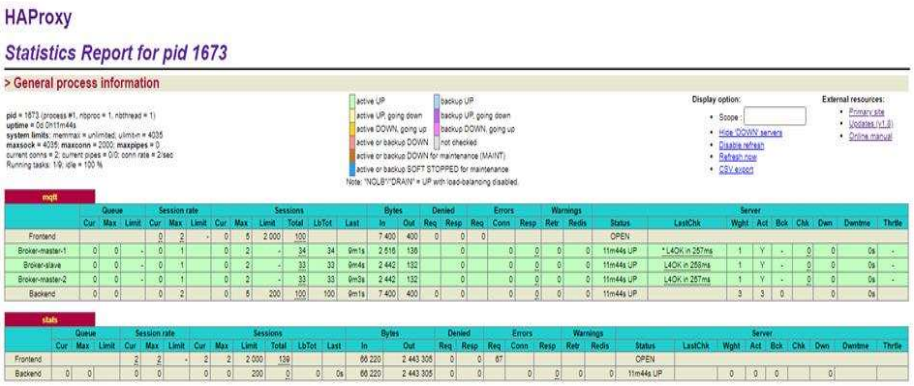
Gambar 3.1.3 *HAProxy configuration load balancer*

Gambar 3.1.3 menggambarkan konfigurasi pada *HAProxy* untuk *MQTT server*. Dimana *HAProxy listen* semua perintah yang masuk ke *port 1883*. Karena pada dasarnya *MQTT* merupakan koneksi *tcp* maka mode menyesuaikan mode *tcp* seperti yang terlihat pada gambar 3.1.3. Algortima yang digunakan adalah *least connection* (memilih *server* dengan jumlah koneksi paling sedikit). Pada baris selanjutnya menentukan kemana *HAProxy* ini akan terhubung, dalam peneltian ini disediakan 3 *broker server* yang dibuat pada *EC2 Instance AWS (Amazon web services)* dengan *IP address* yang berbeda-beda.

```
listen stats
        bind *:1234
        mode http
        stats enable
        stats hide-version
        stats uri /
        stats auth username:password
```

Gambar 3.1.4 *HAProxy statistics*

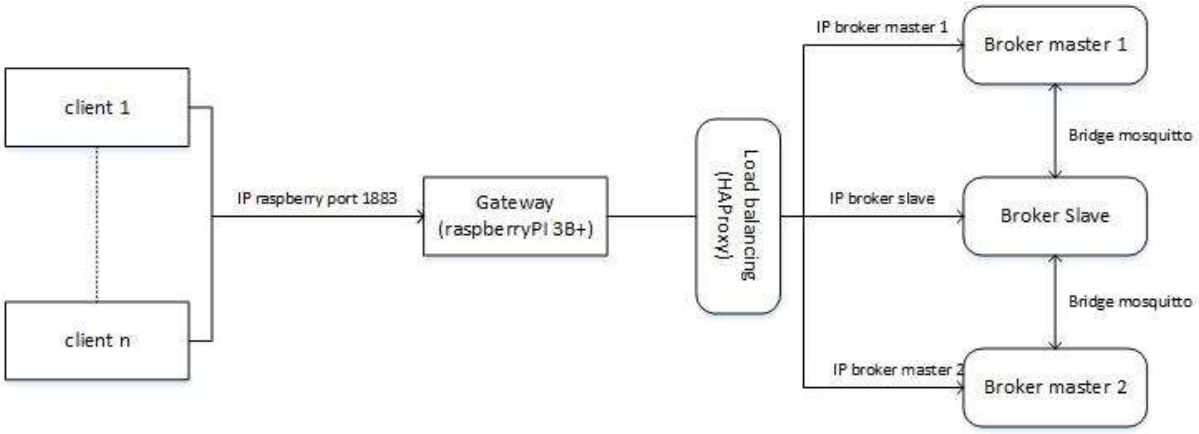
Gambar 3.1.4 menjelaskan konfigurasi pada *HAProxy* untuk *statistics*. *Bind* menjelaskan alamat dan *port* yang akan digunakan untuk mengakses *dashbord*. *Stats auth* digunakan sebagai *autentication* ketika masuk ke halaman *dashbord* dengan *username:username dan password:password*.



Gambar 3.1.5 *Dashbord HAProxy*

Status dan *statistics HAProxy* dapat diakes secara *web base* melalui alamat http://192.168.13.106:1234/stats. Halaman *statistics HAProxy* menjelaskan beberapa informasi untuk melakukan pengecekan kondisi pada *broker server*. Jika server berwarna hijau maka *server* terdaftar dengan status *running* dan tidak mengalami kendala apapun, artinya *broker server* dapat menerima *request client* yang dikirimkan *load balancer HAProxy*. Selain itu *statistics HAProxy* menggambarkan beban yang ditempatkan pada .

3.2 Perancangan gateway



Gambar 3.3.1 Perancangan *gateway*

Gateway memiliki peran sebagai penghubung komunikasi antara *client* dengan *broker server*. Antara *client* dan *gateway* terhubung dalam jaringan lokal *bind* port 1883. Perancangan *gateway* memanfaatkan *raspberrypi 3b+* yang diinstall aplikasi *open source HAProxy* sebagai *load balancer* dengan configurasi tertentu agar dapat mengelola data yang dikirimkan *client* dan dapat diteruskan pada broker server. Ketersedian *gateway* ini penting karena sebagai destinasi pengiriman data yang dikirimkan *client* agar diteruskan ke *broker server*. Dengan demikian ketika *client* ingin mengirimkan pesan atau *request* pesan ke *broker server* harus terhubung terlebih dahulu ke *gateway* dengan mengakses ip *gateway*.

3.3 Perancangan broker server

Selain *publisher* dan *subscriber, broker* juga merupakan komponen penting pada protokol MQTT. *Broker* ini bertugas sebagai jembatan antara *publisher* dan *subscriber*. Karena pada penelitian ini fokus pada *broker*, maka jumlah *broker server* yang disediakan lebih dari satu. Semua *broker server* berjalan pada *operating system Ubuntu 20.04 LTS* yang dijalankan menggunakan *EC2 instance* AWS (*Amazon Web Services*) dan diinstall *mosquitto*. *Broker server* akan melakukan sinkronisasi topic dengan memanfaatkan *bridge mosquitto* agar semua *broker* mendapatkan pesan sama yang dikirimkan oleh *gateway*.

3.4 Analisis kebutuhan

Analisis kebutuhan menentukan kebutuhan apa saja yang diperlukan pada penelitian ini. Sehingga penelitian ini dapat dilakukan untuk mendapatkan hasil penelitian serta dilakukan analisis. Berikut merupakan daftar kebutuhan yang diperlukan.

3.4.1 Kebutuhan perangkat keras

1) Broker server (Virtual)
   Terdapat 3 broker server dalam penelitian ini, dimana semua *broker server* berjalan pada o*perating system ubuntu 20.04 LTS* yang dibuat menggunakan *EC2 instance* AWS (*Amazon Web Services*). Dengan spesifikasi *vCPU* 1, *size memory 1 GB* dan *size storage 8 GB.*
2) Laptop/PC
   Laptop yang digunakan dalam pengerjaan tugas akhir ini adalah *Asus 550v* dengan *processor intel core i7* dan *RAM 8.00GB*. Laptop/pc akan digunakan sebagai *client*.
3) Mikrokomputer raspberryPI 3B+
   Mikrokomputer *raspberryPI 3B+* digunakan sebagai *gatewa*y. Dengan *Operating system NOOBS raspbian. CPU type ARM Cortex-A53 1.4GHz, RAM size 1GB, Integrated Wi-Fi 2.4GHz dan 5GHz dan Ethernet speed 300Mbp*s.

3.4.2 Kebutuhan perangkat lunak

1) Operation system ubuntu 20.04 LTS
   Operation system ini diinstall pada masing-masing *broker server* pada *EC2 instance AWS (Amazon web service).*
2) Operation system windows
3) Operation system NOOBS rasbian
4) Mosquitto
   Software open source yang digunakan sebagai *broker*. *Mosquitto* ini diinstall pada masing-masing *broker server*.
5) MQTTbox
6) Apache jMeter
7) HAProxy
8) Virtualbox
   Digunakan sebagai pembangun *server virtual*.

3.5 Skenario pengujian

Pada penelitian ini dilakukan evaluasi terhadap sistem yang telah dibangun. Tujuannya adalah untuk mendapatkan performansi dari metode *load balancing* algoritma *least connection*. Selain itu dilakukan perbandingan menggunakan algorima *round robin* dan tanpa menggunakan *load balancing*. Pengujian dilakukan dengan bermacam skenario yaitu dengan 100 *client*, 300 *client*, 500 *client*, 700 *clien*t, dan 900 *client* yang mengirimkan pesan secara bersamaan. Dengan skenario tersebut didapatkan nilai parameter uji yaitu *throughput, delay, jitter, error rate*.

1. Tabel 1. Skenario pengujian *load balancing*

Skenario	Algortima *load balacing*	Jumlah *client*	Besar data uji	Waktu	Keterangan
Skenario 1	*Least connection*	100 *client*	1 Kbps	150 *second*	Parameter pengukuran : • *Throughput* • *Delay* • *Jitter* • *Error rate*
Skenario 2		300 *client*		300 *second*	
Skenario 3		500 *client*		1000 *second*	
Skenario 4		700 *client*		1300 *second*	
Skenario 5		900 *client*		1500 *second*	
Skenario 6	*Round robin*	100 *client*	1 Kbps	150 *second*	
Skenario 7		300 *client*		300 *second*	
Skenario 8		500 *client*		1000 *second*	
Skenario 9		700 *client*		1300 *second*	
Skenario 10		900 *client*		1500 *second*	
Skenario 11	1 *broker / no load balancing*	100 *client*	1 Kbps	150 *second*	
Skenario 12		300 *client*		300 *second*	
Skenario 13		500 *client*		1000 *second*	
Skenario 14		700 *client*		1300 *second*	
Skenario 15		900 *client*		1500 *second*	

15 buah skenario jaringan yang berbeda digunakan dalam lingkungan simulasi. Skenario 1, 2, 3, 4, dan 5 digunakan untuk mengkur pengaruh algoritma *least connection* terhadap jaringan load balancing. Skenario 6, 7, 8, 9 dan 10 mengukur pengaruh algoritma round robin terhadap jaringan load balancing. Skenario 11, 12, 13, 14 dan 15 mengukur pengaruh *1 broker* terhadap jaringan load balancing. Dengan cara kerja sebagai berikut:

1. Cara kerja yang dilakukan untuk perbandingan metode *load balancing* algoritma *least connection* dan *round robin* adalah *client* mengirimkan pesan melalui *gateway*, lalu *load balancing HAProxy* akan mengatur pesan dari *client* dengan metode yang diterapkan. Kemudian *broker server* yang menerima pesan akan menyebarkan pesan tersebut ke *broker server* lain yang terhubung sehingga semua *broker* akan menerima pesan yang sama.

2. Cara kerja yang dilakukan apabila *client* mengirimkan pesan tanpa *load balancing* adalah *client* terhubung dengan *broker server* menggunakan ip *broker* tersebut. Pada kasus ini *client* tidak perlu terhubung ke *gateway*.

4. Evaluasi

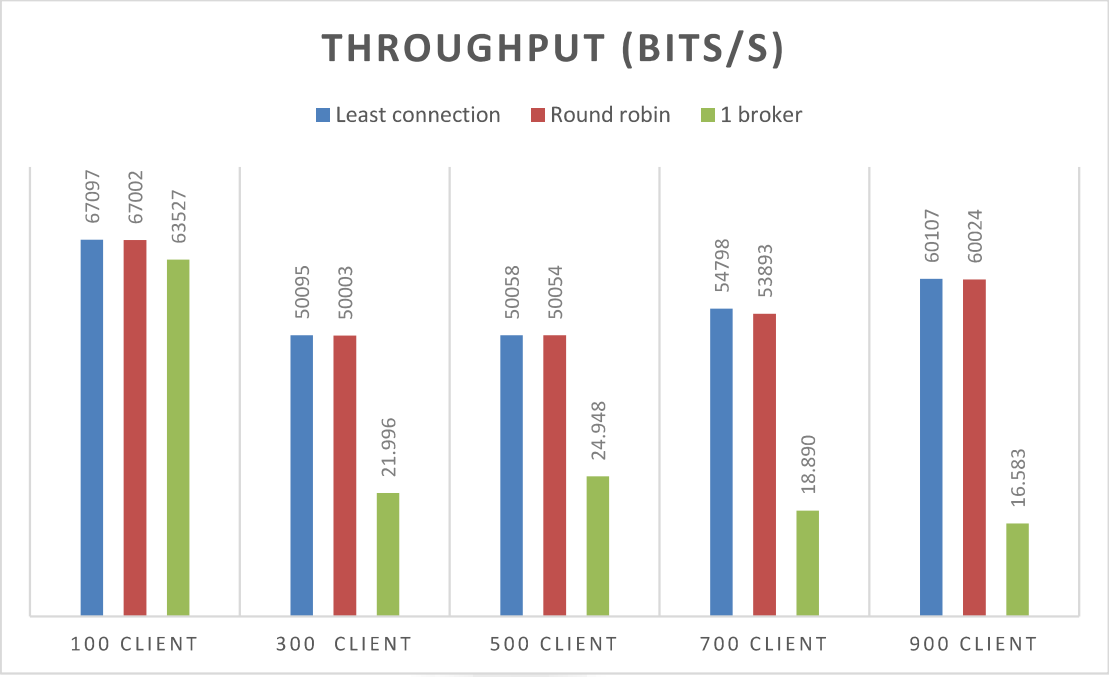
4.1 Hasil Pengujian

Pada perancangan sistem yang dibangun maka didapatkan nilai uji parameter QoS (*Quality of Service*) yaitu *throughput, delay*, *jitter* dan *error rate* dengan 15 kali skenario pengujian yang dilakukan. Karena penelitian ini dilakukan secara simulasi, maka ada beberapa variabel sebagai inputkan yaitu jumlah client, ukuran data uji, run-up periode. Jumlah client adalah banyaknya request pada skenario pengujian. Ukuran data uji adalah besar message yang diinputkan dalam 1 kali periode pengujian. Pada penelitian ini ukuran data yang diinputkan samayaitu 1Kbps. Run-up periode adalah waktu yang disediakan untuk menyelesaikan 1 kali periode pengujian.

Hasil pengujian terhadap parameter Throughput

Througput merupakan tingkat pengiriman pesan yang berhasil melalui saluran komunikasi. Data milik pesan ini dapat dikirimkan melalui tautan fisik atau logis, atau dapat melewati simpul jaringan tertentu . *Throughput* biasanya diukur dalam bit per detik (bit / s atau bps).

$$\textbf{Throughput} = \frac{\textbf{jumlah data yang dikirim}}{\textbf{waktu pengiriman data}}$$



Gambar 4.1.1 *Throughput*

Gambar 4.1.1 merupakan hasil pengujian sistem dengan parameter nilai *throughput* dari *algoritma least connection, round robin* dan tidak menggunakan load balancing/1 *broker*. Pengujian ini dilakukan menggunakan aplikasi *apache jMeter* dengan waktu *run* aplikasi menyesuaian jumlah *client*. Jumlah *client* yang set berbeda-beda yaitu 100 *client*, 300 *client*, 500 *client*, 700 *client* dan 900 *client*. Hal ini dilakukan untuk mendapatkan nilai optimasi perbandingan dari ketiga parameter uji.

Pada penelitian ini nilai *throughput* dipengaruhi oleh jumlah *client*, waktu *running apache jMeter* dan paket yang tidak terkirim (*error rate*). Dengan jumlah *client* dan waktu *running* yang di *set* cukup untuk menyelesaikan *request* dari *client* (*error rate* = 0), menyebabkan tidak terjadinya *error rate* dari masing

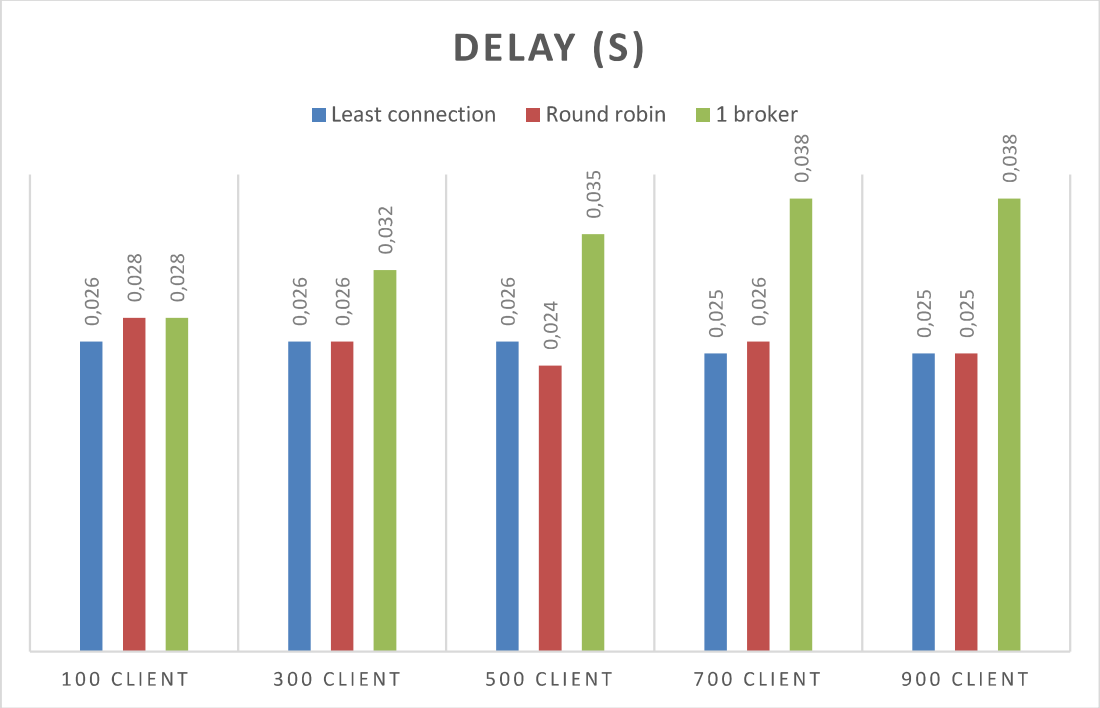
parameter uji. Dengan demikian membuat nilai *throughput* yang didapatkan tinggi. Hal ini tergambarkan pada skenario pengujian 1 yaitu dengan 100 *client*. Dimana nilai *throughput* yang didapatkan lebih tinggi dibandingkan skenario pengujian yang lainnya.

Selain itu semakin banyak jumlah *client* melakukan *request* secara bersamaan dan waktu yang di sediakan tidak cukup untuk menyelesaikan *request* tersebut menyebabkan semakin tingginya nilai *error rate*, berbanding terbalik dengan nilai *throughput* yang di dapatkan semakin rendah. Hal ini tergambarkan pada pengujiaan dengan 1 *broker*, dimana jumlah *client* yang di set bertambah banyak maka nilai *throughput* yang didapatkan menurun. Artinya semakin banyak *client* yang melakukan *request* maka waktu yang dibutuhkan sistem untuk menyelesaikan *request* tersebut juga bertambah agar mengurangi nilai *error* dan meningkatkan nilai *throughput.*

Hasil pengujian terhadap parameter delay

Delay merupakan waktu tunda saat paket yang diakibatkan oleh proses transmisi dari satu titik menuju titik lain yang menjadi tujuannya. *Delay* diperoleh dari selisih waktu pengiriman antara satu paket dengan paket yang lainnya dalam satuan detik.

$$\boldsymbol{Delay = \frac{timespan}{packet}}$$



Gambar 4.1.2 *Delay*

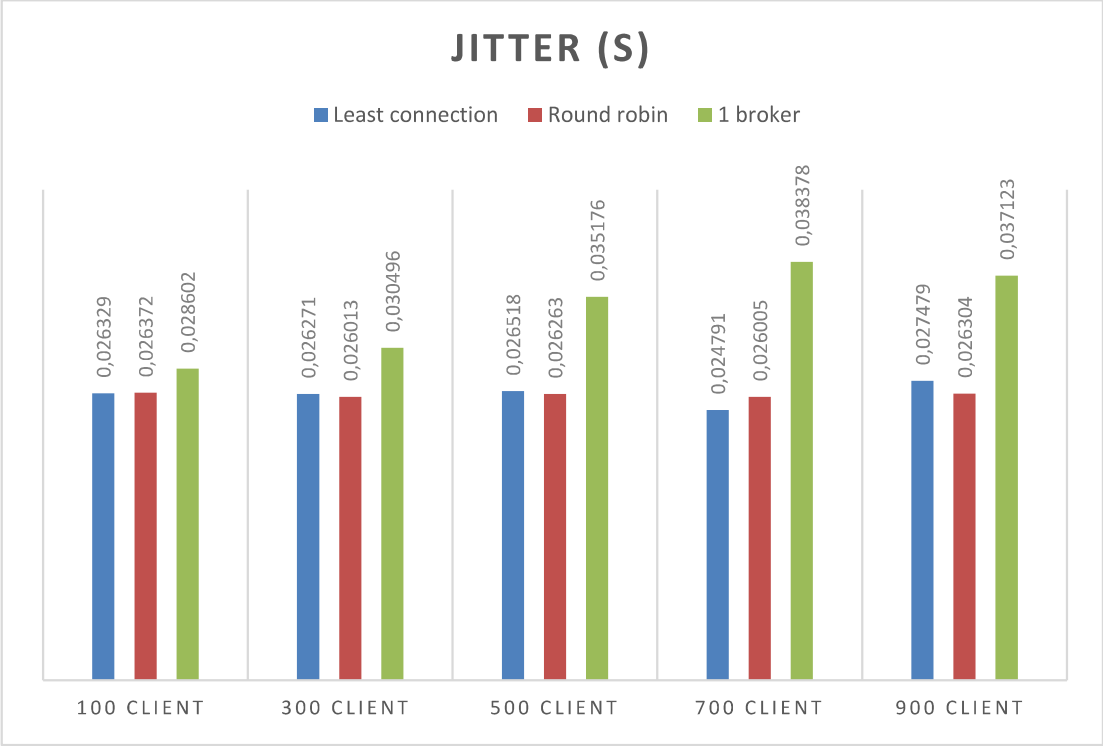
Gambar 4.1.2 merupakan hasil pengujian sistem dengan parameter *delay* dari *algoritma least connection, round robin* dan tidak menggunakan *load balancing*/1 *broker*. Jumlah *client* yang berbeda-beda yaitu 100 *client*, 300 *client*, 500 *client*, 700 *client* dan 900 *client*. Dari skenario pengujian yang dilakukan menggambarkan bahwa *load balancing* membantu *broker server* dalam mengurangi waktu *delay*. Hal ini dikarenakan *load balancing* memastikan ketersediaan dan keadaan yang tinggi dari *server* tersebut. Selain itu *load balancing* dapat meminimalkan *response time* pengiriman paket *request* antara *client* dan *server*. Dari pengujian yang dilakukan yaitu dengan jumlah *client* yang bervariasi dan besar paket data *1Kbps*, terlihat bahwa waktu *delay* dari *load balancing* relatif seimbang yaitu antara 0,024 *second* sampai dengan 0,028 *second.* Namun ketika mengunakan 1 *broker* maka waktu delay yang didapatkan semakin tinggi. Hal ini dikarena ketika menggunakan 1 *broker* maka setiap *client* yang melakukan *request* hanya berfokus ke satu *resource server* tidak adanya pembagian beban ke *broker server* lain.

Hasil pengujian terhadap parameter jitter

Jitter adalah variansi *delay*, yaitu perbedaan selang waktu kedatangan antar paket di terminal tujuan. *Jitter* sangat berhubungan dengan *delay*.

$$Jitter = \frac{total\ variasi\ delay}{total\ paket\ yang\ diterima}$$

$$Total\ variasi\ delay\ = delay - rata\ delay$$

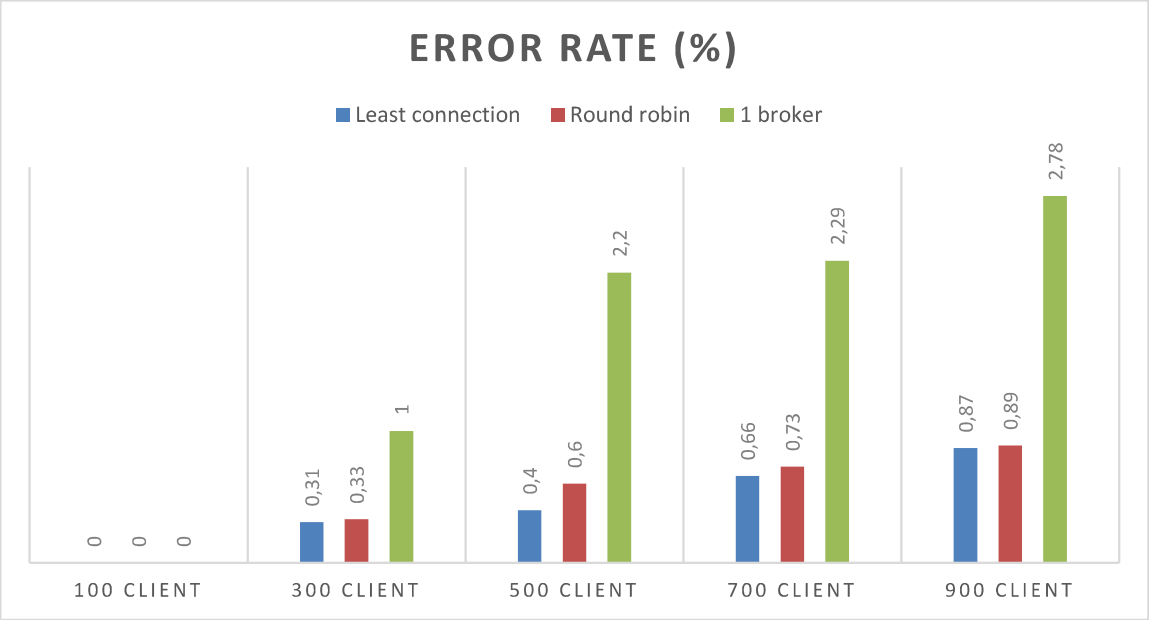


Gambar 4.1.3 *Jitter*

Gambar 4.1.3 merupakan hasil pengujian sistem dengan parameter *jitter* dari *algoritma least connection, round robin* dan tidak menggunakan *load balancing*/1 *broker*. Jumlah *client* yang berbeda-beda yaitu 100 *client*, 300 *client*, 500 *client*, 700 *client* dan 900 *client*. Pengujian ini dilakukan dengan *client* mengirimkan data secara bersamaan antara MQTT *publisher* dan MQTT *subscriber*.

Pada pengujian dengan jumlah *client* 100 didapatkan nilai *jitter* pada algoritma *least connection* sebesar 0,026329*s*, nilai *jitter* pada algoritma *round robin* yaitu 0,026372*s* dan pada 1 *broker* nilai *jitter* didapatkan sebesar 0,028602*s*. Pada pengujian dengan jumlah 300 *client* didapatkan nilai *jitter least connection* yaitu sebesar 0,026271*s*, nilai *jitter* pada *round robin* 0,026013*s* dan pada 1 *broker* didapatkan nilai *jitter* 0,03049*s*. Pada pengujian 500 *client* didapatkan nilai *jitter* pada algoritma *least connection* 0,026518*s*, algoritma *round robin* 0,026263*s* dan dengan menggunakan 1 *broker* yaitu sebesar 0,035176*s*. Pada pengujian dengan 700 *client* didapat nilai *jitter* untuk *least connection* sebesar 0,024791*s*, algoritma *round robin* 0,026005*s* dan dengan 1 *broker* nilai *jitter* sebesar 0,038378*s*. Terakhir pada pengujian dengan 900 *client* nilai parameter *jitter* adalah 0,027479*s,* algoritma *round robin* sebesar 0,026304*s* dan dengan 1 *broker* nilai *jitter* yang didapatkan sebesar 0,037123s.

Hasil pengujian error rate



Gambar 4.1.4 *Error rate*

Berdasarkan gambar 4.1.4 merupakan hasil pengujian sistem dengan parameter *error rate* dari algoritma *least connection, round robin* dan tidak menggunakan *load balancing/1 broker.* Pengujian ini dilakukan menggunakan aplikasi *apache jMeter* dengan waktu *run* aplikasi menyesuaian jumlah *client*. Jumlah *client* yang set berbeda-beda yaitu 100 *client*, 300 *client*, 500 *client*, 700 *client* dan 900 *client*. Hal ini dilakukan untuk mendapatkan nilai optimasi perbandingan dari ketiga parameter uji.

Pada parameter uji ini, jumlah *client* dan *run-up period* yang diberikan untuk menyelesaikan *request* menjadi faktor penyebab terjadinya *error rate*. Pada percobaan dengan jumlah 100 *client* yang melakukan *request* secara bersamaan didapatkan bahwa tidak terjadi *error* saat *client* melakukan *request* pada ketiga paramater uji. Artinya pada skenario pengujian ini sistem dapat menangani banyaknya *request client* dan melakukan distribusi pesan ke *subscriber* 100%.

Pada skenario pengujian berikutnya yaitu dengan 300 *client*, 500 *client*, 700 *client* dan 900 *client* menggambarkan bahwa nilai *error* naik seiring dengan bertambahnya jumlah *client*, semakin banyak *client* yang melakukan *request* secara bersamaan maka nilai *error* semakin tinggi. Dari skenario pengujian algoritma *leat connection* memiliki nilai *error* yang kecil dibandingkan parameter uji lainnya. Hal ini dikarenakan *least connection* memiliki komputasi algoritma *least connection* lebih komplek dengan membandingkan jumlah koneksi di setiap *server*. Selain itu *least connection* cocok digunakan ketika jumlah koneksi besar dengan lalu lintas pengiriman pesan secara merata.

5. Kesimpulan dan Saran

1. Kesimpulan

Dari hasil 15 kali perobaan dengan ukuran data 1 Kbps dan jumlah *client* sesuai dengan grafik diatas maka dapat diambil kesimpulan yaitu:

1. Penerapan metode *load balancing* algoritma *least connection* dapat berjalan dan dapat membantu *broker server* dalam menangani *traffic* yang dikirimkan *client,* dapat dilihat dari hasil percobaan throughput dan error rate, algoritma least connection yang memiliki nilai relatif lebih baik dibandingkan parameter uji lainnya.
2. Semakin banyak *message/request* yang di kirimkan, maka waktu yang dibutuhkan akan semakin banyak. Performansi yang dibutuhkan untuk melakukan *publish message/request* bergantung pada jumlah *message/request*. dimana semakin banyak *message/request* yang di *publish* maka akan semakin tinggi nilai-nilai yang muncul.

3. Dari 5 kali percobaan untuk 5 parameter uji, menggunakan *load balancing* mendapatkan hasil yang rendah dibandingkan dengan menggunakan 1 *broker*, karena *traffic* yang dikirimkan *client* akan dibagi ke *broker server* lain.

2. Saran

Bedasarkan hasil penelitian yang telah dilakukan pada tugas akhir ini, terdapat beberapa saran untuk penelitian selanjutnya:

1. Diharapkan adanya parameter uji lain agar menjadi pembanding nilai optimal pada algoritma *load balancing*.
2. Menambahkan jumlah *client* di setiap pengujiannya.
3. Melakukan pengujian dengan ukuran data yang lebih besar.

Reference

[1] Prasetyo, Bagus, Sabriansyah Rizqika Akbar, and Widhi Yahya. "Implementasi High Availability pada Gateway Wireless Sensor Network dengan Protokol Komunikasi Message Queuing Telemetry Transport." *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN* 2548 (2018): 964X.

[2] Nasser, Husain, and Timotius Witono. "Analisis Algoritma Round Robin, Least Connection, Dan Ratio Pada Load Balancing Menggunakan Opnet Modeler." *Jurnal Informatika* 12.1 (2016).

[3] Nugroho, Agung, Widhi Yahya, and Kasyful Amron. "Analisis Perbandingan Performa Algoritma Round Robin Dan Least Connection Untuk Load Balancing Pada Software Defined Network." *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN* 2548 (2017): 964X.

[4] Charlie, Kevin, Rakhmadhany Primananda, and Mahendra Data. "Analisis Performa Load Balancing Pada Broker MQTT Menggunakan Algoritma Round Robin." *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN* 2548 (2018): 964X.

[5] Permatasari, Elza Fitria Dwi, Aji Gautama Putra, and Maman Abdurohman. "Analisis Perbandingan Performansi Mqtt Dan Http Pada Platform Iot Node-red." *eProceedings of Engineering* 6.2 (2019).

[6] Rakhman, Mohammad Hafidzar, Widhi Yahya, and Kasyful Amron. "Implementasi Metode Failover pada Broker Protokol MQTT Dengan ActiveMQ." *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN* 2548 (2018): 964X.

[7] Ilham, Fitra, Aji Gautama Putrada, and Sidik Prabowo. "Analisis performansi QoS MQTT pada sistem monitoring sungai." *eProceedings of Engineering* 6.1 (2019).

[8] Kusumawardhana, Pramudya Mahardika, Mochammad Hannats Hanafi Ichsan, and Rakhmadhany Primananda. "Implementasi Penyimpanan Data Sensor Nirkabel dengan MongoDB pada Lingkungan IOT Menggunakan Protokol MQTT." *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN* 2548 (2018): 964X.

[9] Apriliansyah, Fahmi, Iskandar Fitri, and Agus Iskandar. "Implementasi Load Balancing Pada Web Server Menggunakan Nginx." *Jurnal Teknologi dan Manajemen Informatika* 6.1 (2020): 18-26.

[10] Jutadhamakorn, Pongnapat, et al. "A scalable and low-cost MQTT broker clustering system." *2017 2nd International Conference on Information Technology (INCIT)*. IEEE, 2017.

[11] Zhu, Liangshuai, Jianming Cui, and Gaofeng Xiong. "Improved dynamic load balancing algorithm based on Least-Connection Scheduling." *2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC)*. IEEE, 2018.

[12] Sahadevan, Arya, et al. "An offline online strategy for IoT using MQTT." *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*. IEEE, 2017.