BAB 1 PENDAHULUAN

1.1 Latar belakang

Internet of Things (IoT) merupakan suatu konsep dimana semua *devices* saling terkoneksi. *IoT* secara sederhana dapat didefinisikan sebagai kondisi ketika *things* atau objek terkoneksi ke internet lebih banyak dibandingkan jumlah manusia (Evans, 2011). Agar setiap *things/devices* bisa saling berkomunikasi, maka dibutuhkan suatu protokol komunikasi pada jaringan internet. Protokol *MQTT* merupakan salah satu protokol yang digunakan untuk komunikasi antar *devices IoT.* Protokol *MQTT* digunakan karena p*rotokol MQTT* merupakan protokol yang ringan. *MQTT* bersifat *open*, simpel, ringan, dan protokol yang mudah diimplementasikan untuk *messaging*. Awalnya, *MQTT* didesain untuk menghubungkan sensor dan kontrol *devices* dalam jumlah yang besar (Tang et al, 2013).

MQTT cocok untuk lingkungan dengan sumber daya yang terbatas dimana setiap *devices* memiliki keterbatasan kemampuan pemrosesan dan memori serta *bandwith* jaringan yang rendah (Dhall & Solanki, 2017). *MQTT* berjalan di atas *TCP* dan menggunakan model *Publish-Subscribe.* Model *Publish-Subscribe* didesain agar mudah untuk diimplementasikan dan bersifat open (Dhall & Solanki, 2017). *Publish-Subscribe* adalah sebuah metode pertukaran pesan secara tidak langsung (*indirect communications*). Pada komunikasi seperti ini, *publisher* membuat *event*, kemudian *event* tersebut dikirim ke *Broker. Broker* berfungsi untuk mengirimkan pesan yang sudah dibuat oleh *publisher* ke *subscriber*, *Broker* hanya akan mengirimkan konten kepada *subscriber* yang sudah melakukan *sub* ke konten-konten tertentu. Dengan adanya *Publish-Subscribe,* seseorang tidak diharuskan menerima semua informasi yang ada. Tetapi informasi yang sesuai dengan kebutuhannya tanpa mengganggu kerja utama sistem yang ada. *MQTT* menggunakan *MQTT Server* sebagai perantara antara *Publisher* dan *Subscriber. MQTT Server* biasa disebut *broker.* Salah satu *broker MQTT* adalah *Mosquitto*.

Mosquitto adalah *message broker* yang *open source* dan sudah mengimplementasikan *MQTT* versi 3.1 dan 3.1.1. *Mosquitto* mendukung *bridge,* yaitu mekanisme agar setiap *broker* bisa saling terhubung dan bertukar informasi pesan *publish*. *Mosquitto* tidak memiliki mekanisme untuk mengatasi kegagalan *server*, sehingga ketika lingkungan yang mengimplementasikan satu *broker Mosquitto* mengalami kegagalan yang disebabkan oleh *overload* pada *broker* ataupun kerusakan pada perangkat keras maka semua *client* yang terhubung ke *broker* tersebut kehilangan akses terhadap data yang dimiliki oleh *broker* tersebut sehingga melumpuhkan fungsi keseluruhan suatu sistem.

Untuk bisa mengirimkan pesan yang hampir menyamai *real time*, *broker* harus selalu terus menjaga koneksi *TCP* yang ada (Hou et al, 2016). Sehingga diperlukan *broker* tambahan yang mampu mengurangi beban kerja yang ada pada *broker.* 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 beban kerja yang ada, maka diperlukan suatu metode bernama *load balancing*. *Load Balancing* adalah teknik untuk mendistribusikan beban trafik pada dua atau lebih jalur koneksi secara seimbang, agar *traffic* dapat berjalan optimal, memaksimalkan *throughput,* memperkecil waktu tanggap dan menghindari *overload* pada salah satu jalur koneksi (Sirajuddin, 2012). Untuk bisa dilakukannya *load balancing,* maka dibutuhkan lah sebuah *device* bernama *load balancer.* Dimana *load balancer* adalah sebuah *device* yang mampu mendistribusikan *traffic* yang ada kemudian mengarahkan paket tersebut ke *server* yang sudah terdaftar dan pendistribusiannya 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).

Kemudian dengan adanya *Load balancer,* kehandalan dan ketersediaan suatu *resource* bisa lebih terjamin, karena ketika terjadi kegagalan pada suatu *server*, maka *load balancer* bisa mengarahkan *traffic* data ke *server* lain yang masih bisa bekerja. Salah satu aplikasi *load balancer* adalah *HAProxy. HAProxy* bisa melakukan distribusi beban kerja berdasarkan 2 jenis paket, yaitu paket *HTTP* dan paket *TCP. Load balancer* memiliki salah satu mekanisme *balancing* menggunakan algoritma *round robin*, algoritma *round robin* dipilih karena algoritma tersebut lebih efisien apabila diterapkan pada *server* dengan spesifikasi yang sama dan beban *load* yang sama (Ardy et al. 2017).

Berangkat dari permasalahan itu, maka diperlukanlah analisis performa *broker Mosquitto* yang memanfaatkan *load balancer* yang menggunakan algoritma *round robin* untuk mengetahui dampak yang diberikan oleh *load balancer* terhadap performa *broker* dalam memproses *request* yang ada. Algoritma *round robin* dipilih karena pada penelitian ini spesifikasi *server* yang digunakan adalah sama, sehingga algoritma *round robin* cocok untuk kondisi ini.

Cara kerja yang ditawarkan adalah *publisher* mengirimkan pesan secara langsung ke semua *broker* yang aktif. Dimana nantinya semua *broker* akan saling bertukar data sehingga tidak ada masalah ketika *load balancer* meneruskan *subscription* ke *broker* yang tidak terhubung dengan *publisher* yang memiliki topik yang sesuai. S*ubscriber* melakukan *subscribe* melalui *load balancer*, dimana *load balancer* akan melakukan distribusi pesan menggunakan algoritma *round robin* untuk meneruskan *request* dari *subscribe*r ke *broker* aktif.

Dengan adanya *load balancing*, beban *broker* tidak hanya terpusat pada 1 titik, selain itu dengan adanya *load balancing*, maka mekanisme rekoneksi ke *broker* lain ketika terjadi kegagalan bisa diimplementasikan. Oleh karena itu dibuatlah penelitian dengan judul "Analisis Performa *Load Balancing* pada *Broker MQTT* Menggunakan Algoritma *Round Robin*".

1.2 Rumusan masalah

Berdasarkan latar belakang yang ada, maka dibuatlah rumusan masalah sebagai berikut :

1. Bagaimana cara menerapkan *load balancing* pada protokol komunikasi *MQTT* khususnya *Broker Mosquitto* ?
2. Bagaimana performa *load balancing* yang menggunakan algoritma *round robin* terhadap CPU *Usage* saat proses *publish-subscribe* ?
3. Bagaimana kecepatan *subscriber* dalam melakukan rekoneksi ketika terjadi kegagalan pada *broker* saat menggunakan *load balancing* dengan algoritma *round robin* ?
4. Bagaimana penggunaan *resource* CPU dan waktu yang dibutuhkan ketika *broker* meneruskan pesan yang diterima dari *publisher* untuk dikirimkan ke *broker* lain yang terhubung pada *bridge*?

1.3 Tujuan

Dari rumusan masalah yang sudah dibuat, maka didapatkan tujuan dari penelitian ini dilaksanakan :

1. Menerapkan *load balancing* pada protokol komunikasi *MQTT* khususnya pada *Broker Mosquitto*.
2. Mengetahui performa *load balancing* yang menggunakan algoritma *round robin* terhadap CPU *Usage* saat proses *publish-subscribe* ?
3. Mengetahui kecepatan *subscriber* dalam melakukan rekoneksi ketika terjadi kegagalan pada *broker*.
4. Mengetahui performa *broker* ketika meneruskan pesan yang diterima dari *publisher* untuk dikirimkan ke *broker* lainnya yang terhubung.

1.4 Manfaat

Manfaat yang diperoleh dari penelitian ini yaitu agar *stakeholder* mampu mengetahui bagaimana *load balancing* mampu meningkatkan *realibility* dari suatu lingkungan, khususnya pada lingkungan dengan sumber daya yang terbatas seperti yang biasa digunakan pada protokol *MQTT*. Kemudian apakah implementasi *load balancer* untuk protokol *MQTT* dapat dilakukan atau tidak. Serta sebagai pertimbangan dalam membuat arsitektur jaringan yang melibatkan protokol *MQTT* yang memanfaatkan *broker* dari *Mosquitto.*

1.5 Batasan masalah

Berdasarkan paparan masalah pada latar belakang masalah, maka batasan masalah yang ditentukan agar cakupan penilitan tidak meluas kemana-mana, maka diperlukan batasan sebagai berikut :

1. Implementasi untuk pengujian hanya berlaku untuk jaringan *Local Area Network*.
2. Analisis yang dilakukan hanya pada kemampuan *load balancer* dan sinkronisasi pesan pada *broker* menggunakan *bridge* terhadap lingkungan sistem.
3. Semua *devices* menggunakan virtualisasi pada 1 *host*.
4. *Mosquitto* berjalan di *foreground* untuk menampilkan *log* pesan.

1.6 Sistematika pembahasan

Penelitian ini akan diuraikan dengan sistematika penulisan yang terbagi menjadi 7 bab sebagai berikut ini :

1. BAB I PENDAHULUAN

   Bab ini berisi latar belakang mengapa masalah ini diangkat, kemudian rumusan masalah yang didapatkan berdasarkan latar belakang masalah, kemudian tujuan penelitian serta manfaat dan sistematika penilitan ini.

2. BAB II LANDASAN KEPUSTAKAAN

   Bab ini berisi dengan teori – teori yang ada serta teori pendukung penelitian yang sedang dibuat. Teori – teori itu bisa berasal dari penelitian sebelumnya maupun studi pustaka dan literatur. Adanya landasan kepustakaan diharapkan mampu menjadi dasar penelitan.

3. BAB III METODOLOGI

   Bab ini berisi tentang metode penilitan yang akan digunakan selama masa penelitian.

4. BAB IV ANALISIS KEBUTUHAN DAN PERANCANGAN

   Bab ini berisi mengenai kebutuhan apa saja yang akan dibutuhkan oleh sistem, tujuan dari sistem, kemudian rancangan sistem yang akan dibuat dan juga skema pengujian yang akan dilakukan kepada sistem untuk mengecek performa yang diberikan oleh sistem.

5. BAB V IMPLEMENTASI

   Bab ini berisi mengenai proses implementasi dari sistem yang ada dan spesifikasi dari setiap komponen yang digunakan untuk proses implementasi.

6. BAB VI PENGUJIAN DAN ANALISIS

   Bab ini berisi mengenai pelaksanaan – pelaksanaan engujian yang sudah dirancang pada bab perancangan. Dimana data hasil pengujian diolah dan ditampilkan dalam bentuk tabel dan grafik untuk dianalisis performanya serta untuk mendapatkan jawaban atas rumusan masalah yang sudah dibuat.

7. BAB VII PENUTUP

Bab ini berisi mengenai kesimpulan yang didapatkan dari pengujian dan pembahasan yang sudah dilakukan. Dimana kesimpulan tersebut menjawab masalah yang sudah dipaparkan pada rumusan masalah. Serta memberikan saran terkait hal-hal yang masih perlu dikembangkan oleh pembaca yang berkeinginan untuk melanjutkan penelitian ini.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Pada penelitian yang berjudul *"Round-Robin Based Load Balancing in Software Defined Networking*" (Kaur et al, 2015) meneliti performa dari setiap *mode balancing* yang ada pada *Software Defined Network*, dimana penelitian ini membandingkan kemampuan algortima *round robin* dengan algoritma *random strategies.* Penelitian ini menyimpulkan bahwa algoritma *round robin* lebih baik jika dibandingkan dengan algoritma *random strategies*. Peneliti menyimpulkan bahwa kelebihan algoritma *round robin* adalah distribusi beban kerja yang adil dan merata. Selain itu, penelitian lain yang berjudul "*Implementasi Load Balancer* Berdasarkan *Server Status* pada *Arsitektur Software Defined Network (SDN)"* (Ardy et al, 2017). Penelitian tersebut juga membahas tentang algoritma *load balancing* pada *Software Defined* Network. Pada penelitian ini peneliti mengajukan suatu teknik *load balancing* berdasarkan *server status* yang menggunakan parameter CPU dan *memory usage* pada jaringan *SDN.* Kemudian metode tersebut dibandingkan dengan teknik *round robin* dan *LBBSRT.* Dari penelitian ini didapatkan hasil bahwa algoritma SS yang ditawarkan peneliti lebih efektif ketika jumlah *request* per detik cukup tinggi untuk *server* dengan spesifikasi yang berbeda (*heterogen)*, kemudian algoritma *round robin* lebih efisien apabila diterapkan pada *server* dengan spesifikasi yang sama (*homogen*) dan beban load yang sama dan algortima *LBBSRT* cenderung lebih cocok pada *server* dengan spesifikasi yang sama akan tetapi beban *load* yang berbeda. Terakhir pada penelitian berjudul "Analisis Perbandingan Performa Algoritma *Round Robin* dan *Least Connection* untuk *Load Balancing* pada *Software Defined Network"* (Nugroho et al, 2017) meneliti performa *load balancing* antara algoritma *round robin* dan *least connection* dengan parameter uji *CPU Usage, response time,* dan *throughput.* Hasilnya algoritma *round robin* memiliki nilai *response time* dan *CPU Usage* yang lebih stabil jika dibandingkan dengan *algoritma least connection.*

2.2 Kajian Teori

2.2.1 *Message Queueing Telemetry Transport* (*MQTT*)

MQTT adalah protokol yang pertama kali dikenalkan oleh Dr Andy Stanford-Clark dan Arlen Nipper pada tahun 1999. *MQTT* merupakan M2M (*Machine to Machine*) protokol. *MQTT* didesain simple, ringan, dan mudah diimplementasikan untuk *messaging*. Karakteristik ini membuat *MQTT* menjadi sangat ideal untuk digunakan pada berbagai situasi termasuk situasi dimana lingkungan dengan sumber daya yang terbatas seperti komunikasi antara *Machine to Machine* dan *Internets of Things* dimana hanya dibutuhkan sedikit baris kode dan *bandwith* jaringan yang bagus adalah suatu kemewahan (Oasis, 2015). *MQTT* dirancang untuk berjalan di atas protokol TCP/IP. Karena menggunakan metode komunikasi *Publish-Subscribe,* maka *MQTT* menggunakan *topic*, dimana *topic* menentukan kemana pesan dari *publisher* akan diteruskan. *Topic* adalah data yang bertipe

string. *MQTT* dirasa tepat untuk menjadi protokol *IoT* karena *MQTT* bersifat light weighted message dan di desain untuk perangkat yang memiliki sumber daya terbatas (Kim, et al, 2015). Gambar 2.1 adalah struktur *Fixed Header* paket *MQTT*.

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type				Flags specific to each MQTT Control Packet type			
byte 2...	Remaining Length							

Gambar 2.1 Struktur *Fixed Header* Paket MQTT (Oasis, 2015)

2.2.2 *Publish-Subscribe* (*Pub-Sub*)

Publish-Subscribe adalah sebuah metode pertukaran pesan secara tidak langsung (*indirect communications*). Pada komunikasi seperti ini, *publisher* membuat *event*, kemudian *event* tersebut dikirim ke *Broker. Broker* berfungsi untuk mengirimkan pesan yang sudah dibuat oleh *publisher* ke *subscriber*, *Broker* hanya akan mengirimkan konten kepada *subscriber* yang sudah melakukan *sub* ke konten-konten tertentu. Semisal *Publisher* melakukan *pub* konten A, maka *Broker* hanya akan memberikan konten A ke *subscriber* yang melakukan *sub* ke konten A. Karena pada metode komunikasi ini tidak ada interaksi secara *end to end*, maka hal ini menyebabkan *Pub-Sub* termasuk ke dalam metode *indirect communications*. Dengan adanya *Publish-Subscribe,* seseorang tidak diharuskan menerima semua informasi yang ada. Tetapi informasi yang sesuai dengan kebutuhannya tanpa mengganggu kerja utama sistem yang ada. Model *Publish-Subscribe* didesain agar mudah untuk diimplementasikan dan bersifat open (Dhall & Solanki, 2017).

2.2.3 *Paho*

Paho MQTT adalah sebuah proyek *open-source* yang menyediakan *library* untuk pengeimplementasian *MQTT*. *Paho MQTT* dikembangkan oleh Eclipse. *Paho MQTT* tersedia dalam beberapa bahasa pemrograman, termasuk Java, C++, C#, .NET, dan *Python*. Selain itu *Paho MQTT* juga memiliki fungsi lain seperti *MQTT*-SN (*Sensor Networks*).

2.2.4 Mosquitto

Mosquitto adalah sebuah proyek *open-source MQTT Server/Broker.* Karena berbasiskan *MQTT*, maka *Mosquitto* sangat cocok untuk digunakan pada *device smartphone, sensor,* dan peralatan *Internet of Things (*IoT*)* yang memiliki kemampuan proses yang rendah (*Mosquitto*.org). Kegunaan *Mosquitto* yaitu sebagai *Broker*. Disini *Broker* berfungsi untuk menyediakan jalur komunikasi antara *publisher* dengan *subscriber*. *Mosquitto* juga sudah memiliki *bridge* yang mengizinkan *broker mosquitto* untuk terhubung dengan *MQTT server* lainnya, termasuk *broker mosquitto* lainnya (Eclipse, 2017).

2.2.5 Load Balancing

Load Balancing adalah teknik untuk mendistribusikan beban trafik pada dua atau lebih jalur koneksi secara seimbang, agar *traffic* dapat berjalan optimal, memaksimalkan *throughput,* memperkecil waktu tanggap dan menghindari *overload* pada salah satu jalur koneksi (Sirajuddin, 2012). Sehingga *load balancing* adalah metode untuk mendistribuskan beban kerja ke beberapa komputer atau ke suatu klaster komputer melalui suatu jalur untuk mencapai penggunaan *resource* yang optimal sehingga didapatkan *throughput* yang tinggi dan mengurangi *response time* yang ada. *Load Balancing* digunakan untuk menghindari terjadinya kelebihan beban pada suatu *resource* dan juga membagikan *traffic* antara *server* dan *data*.

2.2.6 HAProxy

HAProxy adalah sebuah solusi yang gratis, sangat cepat dan handal yang menawarkan *high availability, load balancing,* dan *proxying* untuk aplikasi yang berbasiskan *HTTP* dan *TCP*. *HAProxy* sangat cocok untuk *website* dengan *traffic* jaringan yang tinggi (*HAProxy.org*). Kemudian *HAProxy* sendiri merupakan aplikasi *load balancing* yang ringan dan bersifat *open source*. *HAProxy* menawarkan solusi berupa *load balancing* berdasarkan paket *HTTP* dan pake *TCP* dan berjalan pada model *single thread* dan mendukung konkurensi (Hou et al, 2016).

2.2.7 Algoritma *Round Robin*

Round robin adalah suatu algoritma dimana semua *request* yang diterima akan diteruskan ke *server* selanjutnya yang berada pada daftar. *Round robin* memperlakukan semua *server* secara sama rata tanpa memandang jumlah koneksi yang sedang terjadi pada *server* tersebut. Konsep dasar dari algoritma *Round robin* ini adalah dengan menggunakan *time sharing*, pada intinya algoritma ini memproses antrian secara bergiliran (Ellrod, 2010). Algoritma round robin bekerja dengan cara membagi beban secara bergiliran dan berurutan dari satu server ke server lainnya. Pada penggunaan algoritma *round robin*, saat user mengakses halaman web *server* pertama akan diarahkan ke *server* utama, permintaan *user* yang selanjutnya akan diberikan kepada server yang lainnya (Mustafa & Ibrahim, 2015).