IMPLEMENTASI PROTOKOL MQTT PADA SMART BUILDING BERBASIS OPENMTC

IMPLEMENTASI MQTT PROTOCOL ON SMART BUILDING BASED ON OPENMTC

[1]Gede Okky Satria [2] Gandeva Bayu Satrya, S.T., M.T. [3] Anton Herutomo, ST., M.Eng.
[1 2 3] Fakultas Informatika, Universitas Telkom, Bandung
[1]okkysatria93@gmail.com, [2]gandeva.bayu.s@gmail.com, [3]anton.herutomo @gmail.com

Abstrak

Bangunan adalah salah satu kebutuhan pokok, sebagai ruang yang memberikan kenyamanan dan keamanan bagi penghuninya. Seiring pertumbuhan teknologi dan ilmu pengetahuan, fungsionalitas bangunan pun ditingkatkan. Bangunan yang dapat mengatur konsumsi energinya dan mengurangi dampak pada lingkungan sehingga dapat kita sebutkan sebagai *smart building*.

Hal yang paling mendasar adalah *smart building* dapat memberikan layanan yang lebih seperti pengaturan pencahayaan, keamanan, pengatur suhu dan lain sebagainya. *Smart building* adalah hasil implementasi dari *Machine-to-Machine* (M2M). OpenMTC adalah salah satu *platform* M2M dengan protokol komunikasi yang disediakan adalah HTTP. Pada penelitian ini menerapkan protokol MQTT pada *platform* OpenMTC sebagai protokol komunikasi. Hasil penelitian menunjukan protokol MQTT dapat diterapkan pada *platform* OpenMTC sehingga mendapatkan hasil perbandingan protokol MQTT dan protokol HTTP dengan parameter uji *delay, packet loss, protocol overhead.*

Kata Kunci: ***Smart Building*, M2M, OpenMTC, MQTT, HTTP**

Abstract

Building is one of basic needs, as a space that provides comfortable and safety for occupants. As the growth of technology and science, fungcionality of the building is improved. Building that can regulate energy consumption and reduce the impact on the environment so that it can be mentioned as a smart building.

The most fundamental thing is the smart building can provide more services such as regulating lighting, security, temperature control, etc. smart building is the result of implementation of Machine-to-machine (M2M). OpenMTC is one of M2M platform that provide HTTP protocol as protocol communication. In this research using plartofrm OpenMTC by applying MQTT protocol as the communication protocol. The results showed MQTT protocol can be implemented on the OpenMTC platform so get comparative results MQTT protocol and HTTP protocol by test parameters delay, packet loss and protocol overhead.

***Keywords: Smart Building*, M2M, OpenMTC, MQTT, HTTP**

1 Pendahuluan

1.1 Latar Belakang

Internet adalah salah satu aspek penting dalam sejarah manusia, *Internet of Things* (IoT) dapat dikatakan sebagai evolusi internet yang sangat berpengaruh dalam meningkatkan kualitas hidup manusia. *Internet of Things* (IoT) mengacu pada benda atau perangkat yang terhubung ke internet. Jumlah perangkat yang terhubung ke internet sebesar 12.5 miliar pada tahun 2010 dan populasi penduduk dunia sebanyak 6.8 miliar, perbedaan jumlah perangkat dan penduduk hampir mencapai 2 kali lipat dan diprediksikan akan terus meningkat drastis [1]. Salah satu contohnya menerapkan *smart building* dengan memanfaatkan konsep *Internet of Things* (IoT). Hal yang paling mendasar adalah *smart building* dapat memberikan layanan yang lebih seperti pengaturan pada pencahayaan, keamanan, pengatur suhu dan lain sebagainya. Bangunan masa depan harus memiliki integrasi dari satu benda ke benda lainya dengan tujuan meminimalkan konsumsi listrik dan dampak pada lingkungan sekitar. Bagaimana perangkat dalam bangunan dapat mengumpulkan informasi dan mengontrol benda tersebut dan bagaimana membangun jembatan untuk menyalurkan data yang telah dikumpulkan sehingga tersimpan pada *cloud system* atau *data store.*

Machine-to-machine adalah paradigma dimana komunikasi *end-to-end* dijalankan tanpa adanya intervensi manusia. M2M menghubungkan antar mesin, seperti perangkat, *sensors* dan *actuators.* MQTT adalah protokol komunikasi yang ringan, terbuka dan sederhana, lebih moderen daripada HTTP yang memiliki *protocol overhead* yang rendah, berjalan pada *latency* yang tinggi dan dapat berjalan pada *bandwidth* yang rendah [2].

Pada tugas akhir ini dimaksudkan untuk mendapatkan performansi protokol MQTT yang diimplementasikan pada *platform* M2M yaitu OpenMTC dan membandingkan kinerja protokol MQTT dan HTTP sebagai protokol standar yang disediakan oleh OpenMTC. Pengujian performansi dilakukan dengan tiga parameter uji yaitu *delay, packet loss dan protocol overhead* dan mengunakan dua lingkungan pengujian yaitu menggunakan topologi LAN dan WLAN.

2 Dasar Teori

2.1 Protocol Engineering

Protocol engineering adalah bagian dari disiplin ilmu pada area telekomunikasi yang terdiri dari desain, validasi, dan implementasi protokol komunikasi [3]. Hal yang mendasari protokol komunikasi adalah menggunakan pinsip pengiriman suatu benda dalam kehidupan sehari hari dengan analogi seseorang mengirim pesan. Komunikasi bisa dianalogikan sebagai berikut:

Source (pengirim) – *Trannsmitter* (pengirim ke pos pengiriman) – *Transmission System* (sistem yang mengirimkan pesan seperti dengan mobil, pesawat dll.) – *Receiver* (penerima pesan) – *Destination* (alamat pesan yang dituju) [4].

Analogi komunkasi di atas dapat diserap dan diimplementasikan pada komunikasi elektronik. Komunikasi elektronik membutuhkan suatu protokol komunikasi, protokol adalah sebuah aturan yang menentukan *format*, *frame* dan *packet* dalam pengiriman pesan. Beberapa hal yang perlu diperhatikan dalam *protocol engineering* adalah: [3]

1. Proses pengembangan protokol komunikasi
2. Rancangan protokol
3. Verifikasi protokol
4. Evaluasi kinerja protokol
5. Implementasi protokol
6. Pengujian protokol

Mengukur Protokol Komunikasi

Pengujian protokol adalah metode validasi yang paling penting, mengukur performansi dari protokol dapat dilakukan dengan berbagai jenis, dapat dijelaskan sebagai berikut:

1. *Debugging* [3]
Pengujian ini dilakukan untuk menemukan bugs dalam pengimplementasian protokol.

2. *Conformance test* [3]
Pengujian ini adalah memeriksa apakah implementasi sudah sesuai dengan spesifikasi protokol. Pengujian ini adalah pengujian yang lebih tajam dari *debugging.*

3. *Interoperability test* [3]
Pengujian ini adalah penunjang dari *conformance test*. Ini menguji interaksi antar hasil implementasi protokol, jadi dapat dikatakan pengujian antar *confomance test*. *Conformance test* menghasilkan hasil implementasi pertama, kedua dan ketiga, sehingga *interoperability test* adalah menguji keterhubungan antar *conformance test* pertama, kedua dan ketiga.

4. *Complementary test* [3]
Pengujian ini memiliki dua bagian pengujian yaitu performance test yang mengevaluasi performansi target, seperti *respone time*, *throughput*, dan *robustness test* yaitu memerika protokol masih bekerja stabil ketika diberikan inputan yang salah.

Pada penelitian ini dilakukan Complementary test yaitu pengujian kinerja suatu protokol dengan beberapa aspek penting yang dapat dikalkulasikan yaitu *protocol overhead*, *delay*, dan *packet* l*oss.*

a. *Delay* [5]
Delay adalah penundaan waktu suatu paket yang diakibatkan proses transmisi dari asal ke tujuan. Dalam menghitung *delay* dapat menggunakan persamaan:

$$\text{delay} = \frac{W}{\text{[illegible]}} \quad (2.1)$$

b. *Packet loss* [5]

Packet loss adalah kegagalan transmisi paket dalam mencapai tujuan. Cara menghitung *packet loss* dapat menggunakan persamaan:

$$\text{packet loss} = \frac{\text{[illegible]}}{\text{[illegible]}} \times 100\% \quad (2.2)$$

c. *Protocol Overhead* [6]

Overhead adalah porsi "non-data" dari sebuah protokol. Cara menghitung protocol overhead dapat menggunakan persamaan:

$$\text{[illegible]} = \frac{\text{[illegible]}}{(\text{[illegible]} + \text{[illegible]}} \quad (2.3)$$

2.2 Smart City

Smart city adalah Kota masa depan, dimana seluruh aspek Kota dapat terintegrasi satu dengan lainya. *Smart city* menerapkan jaringan komunikasi dan sensor nirkabel yang bertujuan untuk menyelesaikan masalah perkotaan. Teknologi *smart city* mengintegrasikan dan menganalisis data dalam jumlah yang besar. Beberapa masalah perkotaan yang dapat ditangani dengan menerapkan smart city contohnya mengubah rute lalu lintas (mengurangi kemacetan dan kecelakaan) identifikasi kejahatan, efisiensi konsumsi sumber daya, menemukan

tempat parkir dan lain sebagainya. Beberapa factor yang mempengaruhi perlu diterapkanya *smart city* di berbagai Kota adalah sebagai berikut [6] :

1. Peningkatan populasi : lebih dari 50% populasi dunia tinggal di Kota.
2. Pertumbuhan ekonomi : 600 Kota – Kota besar dunia berkontribusi sebesar 65% dari pertumbuhan GDP sejak 2010 – 2015.
3. Peningkatan gas emisi rumah kaca : memaksa Kota untuk mengembangkan strategi yang berkelanjutan dalam distribusi energi, transportasi, air, perencanaan Kota dan green building.
4. Penurunan anggaran : Anggaran yang besar secara terus menerus dialokasikan untuk menanggapi masalah Kota.

Pembahasan *smart city* tingkat internasional menimbulkan pemahaman dan kesadaran dampak besar mengenai *smart city*, peningkatan penyediaan layanan, kualitas hidup dan ekonomi lokal. Namun dalam penerapan teknologi ini pada kehidupan nyata menemui banyak hambatan dan tantangan. Dalam meneliti tantangan di setiap Kota yang berbeda, penelitian menganalisis enam Kota yang sedang mengarah kepada *smart city*.

2.3 MQTT

Message Queuing Telemetry Transport (MQTT) adalah protokol *transport* yang bersifat *client server publish/subscribe*. Protokol yang ringan, terbuka dan sederhana, dirancang agar mudah diimplementasikan. Karakteristik ini membuat MQTT dapat digunakan di banyak situasi, termasuk penggunaanya dalam komunikasi *machine-to-machine* (M2M) dan *Internet of Things* (IoT). Protokol ini berjalan pada TCP/IP.

Protokol MQTT membutuhkan transportasi yang menjalankan perintah MQTT, *byte stream* dari *client* ke *server* atau *server* ke *client*. Protokol *transport* yang digunakan adalah TCP/IP. TCP/IP dapat digunakan untuk MQTT, selain itu TLS dan WebSocket juga dapat menggunakan TCP/IP. Jaringan yang bersifat *connectionless* seperti *User Datragram Protocol* (UDP) tidak dapat digunakan karena dapat berakibat reorder data [8] .

Berikut merupakan fitur protokol MQTT: [9]

1. *Publish/subscribe message pattern* yang menyediakan distribusi *message* dari satu ke banyak dan *decoupling* aplikasi.
2. *Messagging transport* yang *agnostic* dengan isi dari payload.
3. Menggunakan TCP/IP sebagai konektivitas dasar jaringan.
4. Terdapat tiga level *Qualities of Service* (Qos) dalam penyampaian pesan :
   4.1. "*At most once*", di mana pesan dikirim dengan upaya terbaik dari jaringan TCP/IP. Kehilangan pesan satau terjadi duplikasi dapat terjadi.
   4.2. "*At least once*", dapat dipastikan pesan tersampaikan walaupun duplikasi dapat terjadi.
   4.3. "*Exactly once*", dimana pesan dapat dipastikan tiba tepat satu kali.

2.4 OpenMTC

OpenMTC adalah *platform* untuk mengimplementasikan M2M *middleware* pada *Smart City* dan layanan M2M. OpenMTC dirancang untuk konvergensi *horizontal* yang mendukung kumpulan dari aplikasi *vertical* seperti transportasi dan logistik, eHealth, Utilitas dan lain sebagainya. OpenMTC memberikan dua layanan pada layernya, *Gateway Service Capability Layer* (GSCL) dan *Network Service Capability Layer* (NSCL). Kedua layanan tersebut telah didefinisikan oleh ETSI *Technical Committee* M2M. Berikut kapabilitas dan arsitektur yang dimiliki oleh OpenMTC adalah [10]:

1. *Generic Communication* (xGC) – DGC, GGC, NGC
2. *Application Enablement* (xAE) – DAE, GAE, NAE
3. *Reachability, Addressing and Repository* (xRAR) – DRAR, GRAR, NRAR
4. *Remote Entity Management* (xREM) – DREM, GREM, NREM
5. *Interworking Proxy* (xIP) – DIP, GIP, NIP
6. *Security Capability* (SEC) – DSEC, GSEC, NSEC
7. *Network Communication Selection* (NCS)
8. *Network Telco Operator Exposure* (NTOE)

2.5 Machine-to-Machine (M2M)

Machine-to-Machine (M2M) adalah bentuk komunikasi data yang tidak ada campur tangan manusia. M2M kini lebih berkembang kearah *horizontal system* dan meinggalkan *vertical system*. ETSI *Technical partnership organization* telah menstandari arsitektur dan kapabilitas yang dimiliki *machine-to-machine* (M2M) [11]. Berikut adalah arsitektur dari M2M [11].

2.6 Javascript

Javascript di kembangkan pada tahun 1995 oleh Brendan Eich untuk *Netscape web browser*, Javascript telah terstandari oleh ECMASscript, seringkali dianggap sebagai "*browser language* atau bahasa browser, walaupun mengandung subkata java, javascript sama sekali tidak ada hubunganya dengan pemrograman Java. Tipe data yang dimiliki adalah *numbers, Strings, Arrays, Object, Functions* [12].

Beberapa kapabilitas yang dimiliki oleh javascript adalah [13]:

1. *Lightweight*
2. Dirancang untuk membuat aplikasi *network-centric*

3. Terintegrasi dengan Java
4. Terintegrasi dengan HTML
5. Terbuka dan *cross-platform*

2.7 Node Js

Node.js adalah *platform* yang dibangun di Chrome's *Javascript runtime* untuk memudahkan pembangunan yang cepat dan aplikasi jaringan yang *scalable*. Node.js menggunakan *event-driven*, model *non-blocking* I/O yang membuatnya ringan dan efisien, baik digunakan untuk aplikasi *real-time* yang berjalan diseluruh perangkat yang didistribusikanya [14]. Node.js sebagai *server-side platform* yang dirancang dengan *library* yang minimalis. Ini berjalan di atas V8 *Javascript engine* [15].

Node.js memiliki kelebihan terkait membangun aplikasi jaringan. Node.js mampu mengani masalah I/O bound. Masalah I/O bound dapat diatasi dengan peningkatan *throughput* I/O seperti *disk*, *memori*, *bandwidth* jaringan, dan meningkatkan *caching* data. Ketika ada masalah bagaimana mengani sepuluh ribu atau koneksi secara bersamaan ke server web, beberapa platform akan memputuhkan berbagai *patch* dan penangananny, tapi itu tidak perlu dilakukan jika menggunakan node.js, karena node.js didasarkan pada *non -blocking*, arsitektur yang asinkron dirancang untuk *concurrency*. [15].

2.8 JSON

JSON (*JavaScript Object Notation*) adalah format pertukaran data yang yang ringan. Mudah dibaca dan ditulis oleh manusia, serta mudah diterjemahkan dan dibuat (*generate*) oleh computer. Format ini dibuat berdasarkan bagian dari bahasa emrograman JavaScript, standar ECMA-262 Edisi ke-3-Desember 1999. JSON adalah format teks yang tidak bergantuk pada bahasa pemrograman apapun karena menggunakan bahasa yg umum digunakan oleh programmer keluarga C, seperti C, C++ , C#, Java, JavaScript, Perl, Python dan lain sebagainya. Oleh karena itu JSON merupakan format teks yang ideal sebagai bahasa pertukaran data.

JSON terbuat dari dua struktur:

1. Kumpulan pasangan nama/nilai. Pada beberapa bahasa, hal ini dinyatakan sebagai objek *(Object),* rekaman *(record),* struktur *(struct),* kamus *(dictionary),* tabel hash *(hash tabel),* daftar berkunci *(keyed list)*, atau *associative array*.
2. Daftar nilai terurut *(ordered list of values).* Pada kebanyakan bahasa, hal ini dinyatakan sebagai larik *(array),* vector *(vector),* daftar *(list),* atau urutan *(sequence).*
   Struktur-struktur data ini disebut sebagai struktur data universal. Pada dasarnya, semua bahasa pemrograman moderen mendukung struktur data ini, dalam bentuk yang serupa atau berbeda [16].

2.9 RESTful Web Service

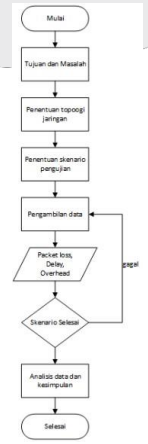
Sebuah *web service* adalah sebuah halaman web yang memberi layanan untuk computer dapat memberikan *request* dan proses, atau lebih tepatnya *web service* adalah halaman web yang dimaksudkan untuk memberikan layanan dari *web browser* lawan atau *tool* UI lainya. *Web service* memerlukan suatu arsitektur yang mendukung layanan, karena tidak ada kecerdasan manusia yang dapat tetap memantau kondisi pada *client end.*

REST adalah *Representational State Transfer*, sebuah *architectural style* dan bukan sebuah *toolkit*. Prinsip – prinsip berikut yang mendukung aplikasi RESTful bisa menjadi sederhana, ringan dan cepat adalah sebagai berikut:

1. Identifikasi sumber daya melalui URI.
2. *Uniform Interface*.
3. *Self-descrptive messages.*
4. *Hypertext* adalah *engine* dari *application state*.

3 Perancangan dan Implementasi

Pada Bab ini membahas mengenai proses implementasi protokol MQTT pada OpenMTC. Implementasi system melewati beberapa tahap. Berikut adalah flowchart yang dirancang pada tugas akhir ini:



Gambar 3.1 Flowchart pengerjaan tugas akhir

Pada tugas akhir ini dibuat sebuah sistem komunikasi *smart building* menerapkan protokol MQTT pada OpenMTC. Protokol MQTT merupakan protokol komunikasi sebagai pembanding protokol standar OpenMTC yaitu protokol HTTP. Pada penelitian ini menggunakan sensor virtual.

Rancangan sistem yang dibangun terdiri dari dua bagian, yaitu OpenMTC dan *Sensor Network*. *Proxy* digunakan untuk menjembatani antara protokol MQTT masuk ke dalam OpenMTC. *Proxy* diletakan pada komputer *server* yang telah ter-*install* OpenMTC. Pada bagian OpenMTC, terdiri dari *application* dan *service capability*. *Application* berfungsi menampilkan *service* dan data yang disediakan oleh OpenMTC. Sedangkan *service capability* menyediakan berbagai layanan komunikasi.

Pada penelitian ini mengimplementasikan protokol MQTT dengan mengirim data *dummy* dengan sensor virtual dan dikirimkan ke komputer *server* yang ter-*install* OpenMTC.

Aliran data dimulai dari *sensor network* yang mengirimkan data dan diterima oleh *proxy* dan diteruskan GSCL pada OpenMTC yang nantinya data tersebut disimpan pada *database* (MongoDB).

Sensor virtual digunakan karena adanya keterbatasan perangkat sensor yang tersedia serta jeda pengiriman dan besar data tiap sensor tidak menentu dapat berpengaruh pada hasil pengujian sehingga kurang baik dalam membandingkan hasil pengujian antara protokol HTTP dan MQTT. Sensor virtual yang digunakan adalah sensor suhu dan sensor cahaya yang merepresentasikan besar suhu dan kondisi cahaya (lampu).

4 Pengujian dan Analisis Hasil Implementasi

Pengujian dilakukan dengan mengirimkan data dari *Device App* sebagai sensor virtual ke komputer server yang terinstall OpenMTC. Pengiriman dilakukan secara berkala dengan jumlah sensor yang berbeda serta ifrastruktur jaringan yang berbeda. Kemudian dilakukan pengukuran kinerja oleh paket data yang dikirimkan dari *Device App* menuju komputer server dengan parameter *delay*, *overhead* dan *packet loss.*

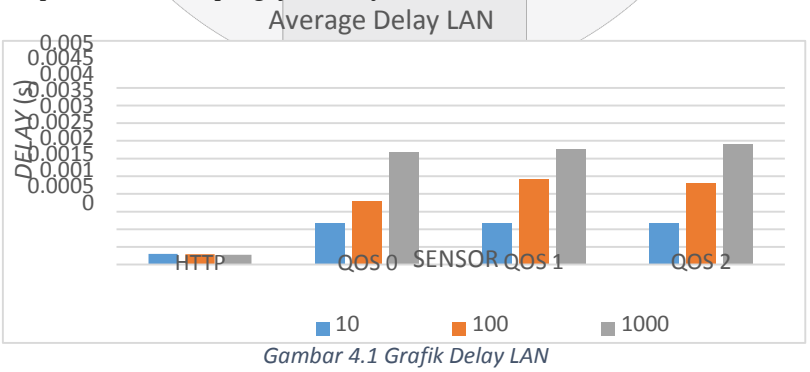
Skenario Pengujian terdiri dari enam macam dengan perbedaan pada jumlah sensor dan ifrastruktur jaringan yang digunakan. Berikut skenario pengujian:

*Tabel **Error! No text of specified style in document.**-1 Skenario Pengujian*

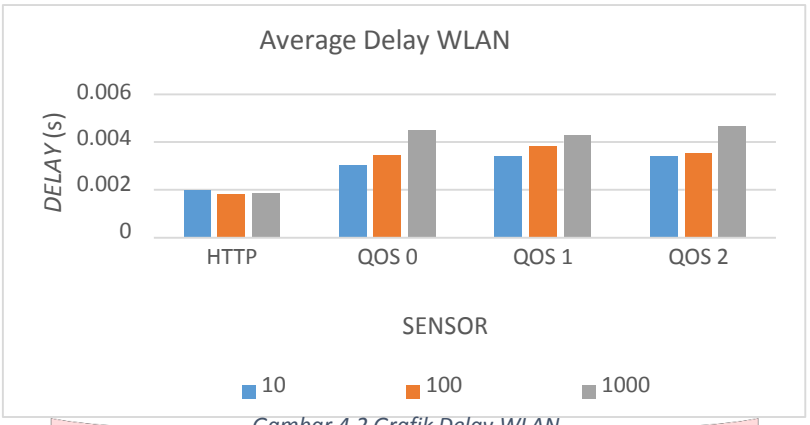
Skenario	Protokol	Jumlah Sensor	Topologi	Parameter
1	HTTP	10	LAN	Delay Packet Loss Overhead
2	HTTP	100	LAN	
3	HTTP	1000	LAN	
4	MQTT	10	LAN	Delay Packet Loss Overhead
5	MQTT	100	LAN	
6	MQTT	1000	LAN	
7	HTTP	10	WLAN	Delay Packet Loss Overhead
8	HTTP	100	WLAN	
9	HTTP	1000	WLAN	
10	MQTT	10	WLAN	Delay Packet Loss Overhead
11	MQTT	100	WLAN	
12	MQTT	1000	WLAN	

4.1 Analisis Pengujian Delay

Berikut merupakan hasil dari pengujian delay:



Gambar 4.1 Grafik Delay LAN



Gambar 4.2 Grafik Delay WLAN

Pengujian *Delay* pada protokol MQTT lebih besar diakibatkan jalur pengiriman data yang berbeda. Protokol HTTP mengirim data dari virtual sensor diterima langsung oleh *server* OpenMTC, sedangkan protokol MQTT mengirim data dari virtual sensor melalui *proxy server*, kemudian melanjutkan paket data atau *forward* dengan format data HTTP dan diterima oleh *server* openMTC sehingga mengakibatkan *delay* MQTT lebih besar dari HTTP.

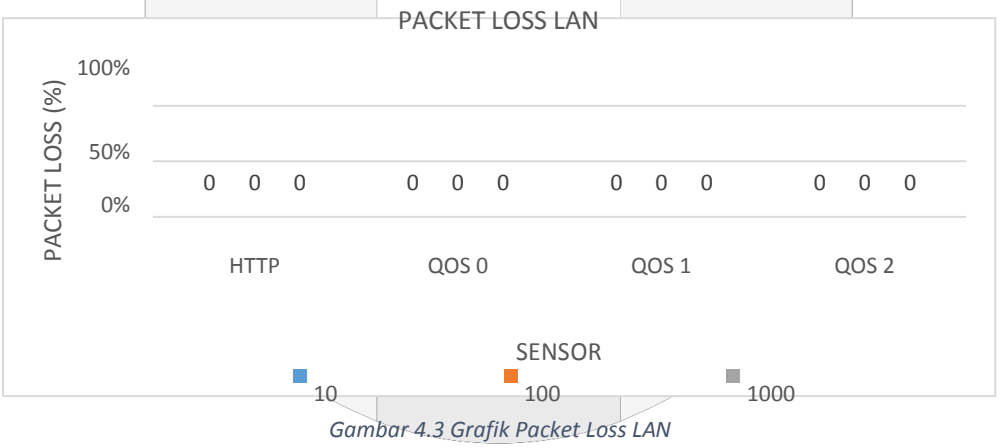
Rata rata *delay MQTT* meningkat seiring meningkatnya jumlah sensor, ini dikarenakan saat MQTT mengirim data melalui *proxy* menuju *server* OpenMTC. Dari virtual sensor menuju *proxy*, *proxy* melakukan *get* MQTT dan dari proxy menuju GSCL, proxy mengirim pesan *request* dan menunggu *response* untuk merubah format data dari MQTT ke HTTP.

Delay terbesar terdapat pada pengiriman data dengan protokol MQTT dan menggunakan topologi WLAN dikarenakan pada wireless kecepatan dapat turun seiring dengan meningkatnya jumlah *client* yang terhubung, mobilitas jaringan dan kuat sinyal yang didapat.

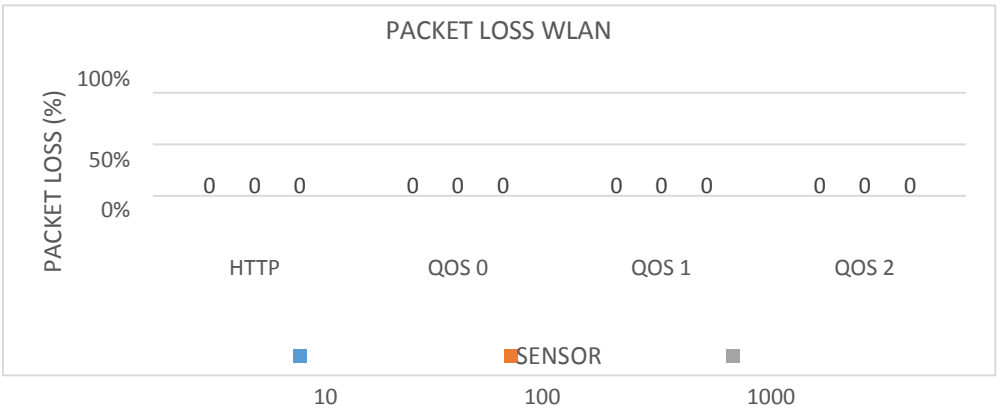
Berdasarkan hasil pengujian, *average delay* pengiriman data pada protokol HTTP adalah lebih baik dibanding protokol MQTT.

4.2 Analisis Pengujian Packet Loss

Berikut merupakan hasil dari pengujian packet loss:



Gambar 4.3 Grafik Packet Loss LAN

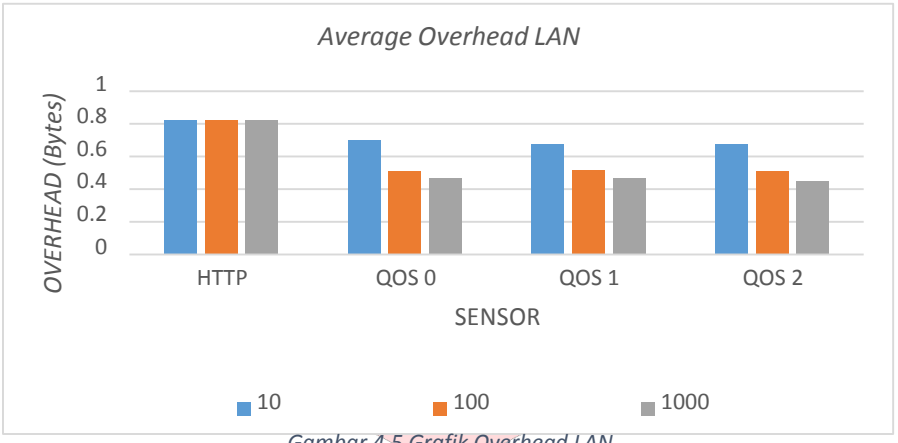


Gambar 4.4 Grafik Packet Loss WLAN

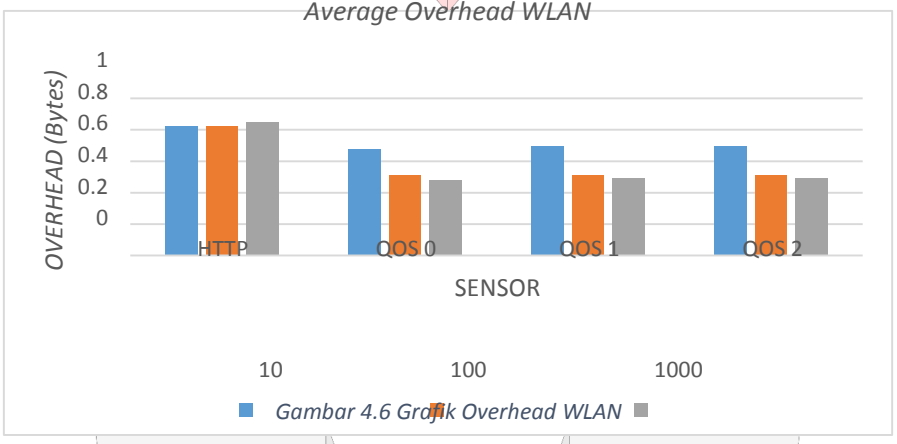
Pengujian *Packet loss* menghasilkan 0%. *Packet loss* biasanya disebabkan oleh kepadatan jaringan. Karena lingkungan pengujian menggunakan LAN dan WLAN sehingga kondisi jaringan dapat dipantau maka dapat disimpulkan pengiriman data dari virtual sensor menuju *server* OpenMTC pada lingkungan pengujian ini akurasinya adalah 100%.

4.3 Analisis Pengujian Overhead

Berikut merupakan hasil dari pengujian overhead:



Gambar 4.5 Grafik Overhead LAN



Gambar 4.6 Grafik Overhead WLAN

Pengujian *Protocol overhead* pada protokol HTTP lebih besar diakibatkan adanya aktifitas SYN ACK yang dikirim secara terus menerus, sedangkan MQTT mengirimkan *request* CONNACK yang berarti ACK untuk *connection request* dan hanya dilakukan di awal saja, selanjutnya dilakukan dengan PUBLISH untuk mengirimkan paket data, selain itu *control packet* pada MQTT memiliki *fixed header* sebesar 2 byte. Pada protokol MQTT terdapat perbedaan rata rata *protocol overhead* pada jumlah sensor 10, 100, dan 1000. Rata rata *protocol overhead* menurun seiring meningkatnya jumlah sensor, ini dikarenakan saat MQTT mengirim *request* CONNACK dan komunikasi telah *established*, *publisher* hanya mengirimkan *message data* saja tanpa *header* sehingga keseluruhan data terdapat dalam *TCP stream* yang sama.

Metode pengiriman data *(request/response* dan *publish subscribe*) dan jumlah sensor pada MQTT mempengaruhi besarnya *protocol overhead* namun menggunakan jenis koneksi yang berbeda tidak memberikan perbedaan yang signifikan antara LAN dan WLAN.

Berdasarkan hasil pengujian dapat disimpulkan bahwa *protocol overhead* pada protokol MQTT adalah lebih baik dibanding protokol HTTP.

5 Kesimpulan dan Saran

5.1 Kesimpulan

Berdasarkan pengujian dan analisis kinerja protokol MQTT pada *Smart Building* berbasis *platform* OpenMTC, maka diperoleh kesimpulan:

1. Protokol MQTT dapat diimplementasikan pada *platform* M2M OpenMTC
2. Pengujian implementasi protokol MQTT pada OpenMTC dengan parameter *delay* dihasilkan *average delay* terbesar terdapat pada protokol MQTT dengan topologi WLAN dan jumlah sensor pada protokol

MQTT mempengaruhi besarnya *average delay*. Dapat disimpulkan bahwa *delay* pengiriman data pada protokol HTTP adalah lebih baik dibanding protokol MQTT.

3. Pengujian implementasi protokol MQTT pada OpenMTC dengan parameter *packet loss* dari virtual sensor ke *server* OpenMTC dengan protokol HTTP dan MQTT yaitu 0% dengan topologi LAN dan WLAN sehingga dapat disimpulkan akurasi pengiriman data sensor pada lingkungan pengujian ini adalah 100%.
4. Pengujian implementasi protokol MQTT pada OpenMTC dengan parameter *protocol overhead* dihasilkan *average protocol overhead* pada protokol HTTP lebih besar dibanding dengan protokol MQTT dengan topologi LAN dan WLAN. Berdasarkan hasil pengujian dapat disimpulkan bahwa *protocol overhead* pada protokol MQTT adalah lebih baik dibanding protokol HTTP.

5.2 Saran

Berdasarkan penelitian Tugas Akhir ini, penulis memiliki saran untuk pengembangan selanjutnya, yaitu:

1. Mengimplementasikan protokol MQTT pada GSCL dan NSCL serta dapat merubah protokol komunikasi pada OpenMTC menjadi MQTT sehingga pengujian *delay* dapat dilakukan dengan lebih baik.
2. Mengimplementasikan pada topologi jaringan yang lebih kompleks.

Daftar Pustaka

[1] D. Evans, "The Internet of Things," *How the Next Evolution of the Internet,* 2011.

[2] J. Vermullard, "M2M, IoT, DEVICE MANAGEMENT," *ONE PROTOCOL TO RULE THEM,* 2014.

[3] H. König, "Protocol Engineering," 2012.

[4] P. P. Venkataram, "Introduction to Basics of Communication Protocol".

[5] M. Rayhan Yuvandra, "ANALISIS KINERJA TRAFIK VIDEO CHATTING PADA SISTEM CLIENT-CLIENT DENGAN APLIKASI WIRESHARK".

[6] G. F. &. S. Mitchell, "Smart City Framework," *A Systematic Process for Enabling Smart+Connected Communities,* 2012.

[7] D. f. B. I. &. Skills, "International case Studies on Smart Cities," 2013.

[8] R. J. Cohn, "MQTT Version 3.1.1," 2014.

[9] D. Locke, "MQ Telemetry Transport (MQTT) V3.1 Protocol," 2010.

[10] F. FOKUS, "OpenMTC Platform A Generic M2M Communication Platform," 2012.

[11] F. FOKUS, "The OpenMTC framework: platform communication,data, interfaces, APIs, policies, interworking," 2013.

[12] F. FOKUS, "Implementation and administration of M2M applications," 2013.

[13] T. Point, "Javascript tutorial".

[14] i. Joyent. [Online]. Available: http://nodejs.org/.

[15] D. Nguyen, Jump Start Node JS, Collingwood, 2012.

[16] Json, "json.org," [Online].

[17] J. Cowan, "RESTful Web Services," *An introduction to building Web Services without tears (i.e., without SOAP or WSDL),* 2005.