Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 2, No. 10, Oktober 2018, hlm. 3508-3514 http://j-ptiik.ub.ac.id

Implementasi Metode *Failover* pada *Broker* Protokol MQTT Dengan *ActiveMQ*

Mohammad Hafidzar Rakhman[1], Widhi Yahya[2], Kasyful Amron[3]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]haipijaaar@icloud.com, [2]widhi.yahya@ub.ac.id, [3]kasyful@ub.ac.id

Abstrak

Internet of Things (IoT) dapat diartikan sebagai konsep untuk menghubungkan "*Things*" atau benda-benda nyata di dunia dengan *internet* dan saling berbagi informasi. Untuk mengirimkan informasi antar *"Things"* melalui *internet*, Iot membutuhkan protokol komunikasi yang ideal dan efisien dalam hal *resource* ketika mengirimkan pesan. Salah satu protokol yang dianggap cocok dengan model komunikasi IoT adalah protokol *Message Queueing Telemetry Transport* (MQTT). Pada model komunikasi protokol MQTT, *broker* memiliki peranan penting dalam keberhasilan proses komunikasi yang terjadi. Ketika *broker* terhenti, *publisher* dan *subscriber* tidak dapat melakukan proses komunikasi dan harus menunggu untuk *broker* diperbaiki. Solusi dari permasalahan pada *broker* dapat diminimalisir dengan mengimplementasikan *failover* antara *master* dan *slave broker*. Oleh karena itu, penelitian ini mengimplementasikan *failover* untuk memperbaiki ketersediaan layanan pada *broker*. Berdasarkan hasil pengujian, sistem memperoleh rata-rata nilai *downtime* sebesar 24,3266 detik dari 5 pengujian. Sementara pada pengujian performa, sistem mendapat rata-rata nilai *latency* sebesar 1,1763 detik pada sisi *publisher* dan 0.1157 detik pada sisi MQTT *subscriber*. Pada pengujian packet loss, pesan yang hilang berjumlah 20 pesan dari 50 pesan ketika mengimplementasikan *failover*.

Kata kunci: *message queueing telemetry transport (MQTT), failover, availability, activeMQ*

Abstract

Internet of Things (IoT) can be interpreted as a concept to connect "Things" or real objects in the world with the internet and sharing information. To share information between "Things" over the internet, Iot needs an ideal and efficient communication protocol in terms of resources when sending messages. One of the protocols considered to fit with IoT communication model is the Message Queuing Telemetry Transport (MQTT) protocol. In the MQTT protocol communication model, the broker has an important role in the success of the communication process that occurs. When the broker stops, the publisher and subscriber can't make the communication process and have to wait for the broker to be fixed. The solution of broker problem can be minimized by implementing failover between master and slave broker. Therefore, this study implements failover to improve the availability of services to brokers. Based on the test results, the system obtained an average downtime value of 24.3266 seconds from 5 tests. While on performance testing, the system gets an average latency value of 1.1763 seconds on the publisher side and 0.1157 seconds on the MQTT subscriber side. In packet loss testing, missing messages amount to 20 messages from 50 messages when implementing failoverl.

Keywords: *message queueing telemetry transport (MQTT), failover, availability, activeMQ*

1. PENDAHULUAN

Teknologi informasi dan komunikasi telah berkembang sangat pesat dan menghadirkan suatu konsep yang dinamakan dengan *Internet of Things* (IoT). IoT dapat diartikan sebagai konsep untuk menghubungkan "*Things*" atau benda-benda nyata di dunia dengan *internet* dan saling berbagi informasi. Iot mengubah benda-benda fisik di dunia dari benda tradisional menjadi *smart* (Al-Fuqaha, et al., 2015). Konsep IoT memungkinkan benda-benda fisik untuk melihat, mendengar, berpikir, berbagi informasi dan mengkoordinasikan keputusan. Iot diharapkan dapat berkontribusi pada peningkatan kualitas hidup masyarakat luas

seperti contohnya pada *smart-homes*.

Untuk mengirimkan informasi antar *"Things"* melalui *internet*, Iot membutuhkan protokol komunikasi yang ideal dan efisien dalam hal *resource* ketika mengirimkan pesan. Hal itu disebabkan karena pada umumnya Iot dibangun dengan *constrained devices* yang memiliki keterbatasan dalam hal *memory*, *storage* dan *power* (Petersen, Bacelli & Wahlisch, 2014). Salah satu protokol yang dianggap cocok dengan model komunikasi IoT adalah protokol *Message Queueing Telemetry Transport* (MQTT). Protokol MQTT mengungguli *Hypertext Transfer Protocol* (HTTP) dalam komunikasi IoT dikarenakan memiliki *overhead protocol* yang lebih rendah dan lebih efisien dalam hal penggunaan *bandwidth* serta *network resources* ketika berkomunikasi (Yokotani & Sasaki, 2016). Hal itu disebabkan karena protokol MQTT adalah protokol yang secara spesifik didesain untuk *devices* yang memiliki keterbatasan *resources* dengan *high-latency* dan *low bandwidth* (Happ & Wolisz, 2016).

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 topik.

Fokus penelitian ini adalah menanggulangi permasalahan terhentinya komunikasi antara *publisher* dan *subscriber* dengan *broker* dalam waktu yang lama ketika *broker* mengalami kegagalan. Mekanisme yang dapat digunakan adalah dengan mengimplementasikan *failover* pada *broker*. *Failover* atau bisa disebut dengan *High-Availability clusters* adalah kumpulan dari beberapa komputer yang mendukung aplikasi *server* dengan *downtime* yang minimal karena adanya grup yang terdiri dari beberapa *server* (Kahanwal & Singh, 2012). *Failover* dapat meminimalisir *downtime* yang terjadi karena adanya komponen lain dalam grup akan menggantikan komponen yang gagal secara otomatis tanpa adanya intervensi dari *user*.

Untuk dapat mengimplementasikan *failover*, *broker* dari MQTT harus menggunakan *message broker* yang memiliki fitur *broker cluster*. Hal itu disebabkan karena *failover* bekerja dengan membentuk sebuah *cluster* yang terdiri dari minimal 2 *node* (Pribadi, 2013). *Cluster* digunakan agar dapat menggunakan redundansi yang mendeteksi kegagalan pada salah satu *node* dalam *cluster* dan menggantikan perannya dengan *node* lain dalam *cluster*. Salah satu *message broker* yang memiliki fitur *broker cluster* adalah *Apache ActiveMQ*. *ActiveMQ* adalah *open source message broker* yang dimiliki oleh *Apache Software Foundation*. *ActiveMQ* memiliki kemampuan konfigurasi *clustering* yang dinamakan dengan *network of brokers* seperti contohnya pada konfigurasi *master-slave* (Magnoni, 2015). Oleh karena itu, penelitian ini menggunakan *message broker ActiveMQ* karena memiliki fitur *broker cluster* yang dibutuhkan agar dapat mengimplementasikan *failover*.

Beberapa peneliti telah melakukan penelitian terkait dengan *failover* untuk mengatasi permasalahan *availability* suatu sistem. Penelitian dari Pribadi (2013) membahas mengenai ketersediaan data dan layanan pada perusahaan sangat dibutuhkan dalam mendukung proses bisnis. Berdasarkan hasil penelitian, implementasi *failover* memastikan bahwa layanan tetap tersedia dengan adanya *primary* dan *secondary server*. Penelitian selanjutnya membahas tentang implementasi *failover clustering* pada *web server* untuk mengatasi permasalahan ketersediaan layanan suatu *website* yang dibutuhkan untuk memberikan informasi. Implementasi dilakukan dengan membangun *cluster* yang terdiri dari *server* aktif dan pasif yang mengimplementasikan *failover* untuk mengatasi kegagalan pada *server*. Berdasarkan hasil penelitian, implementasi *failover* dengan *cluster* dapat meminimalisir *downtime* yang dihasilkan yaitu 5 detik (Juliharta, Supedana & Hostiadi, 2015). Berdasarkan pada beberapa penelitian sebelumnya, *failover* dapat dijadikan solusi untuk mengatasi permasalahan ketersediaan layanan suatu sistem dengan adanya 2 *node* yang meminimalisir waktu *downtime*.

Berdasarkan latar belakang yang diuraikan pada paragraf sebelumnya, maka penelitian ini membangun sebuah sistem dari protokol MQTT yang mengimplementasikan *failover* dengan

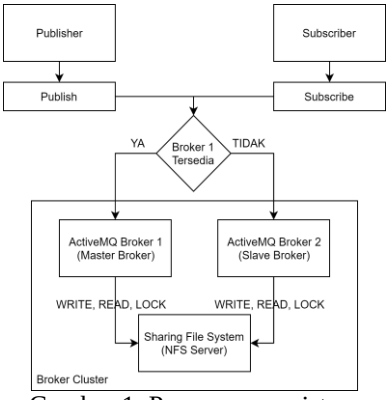
membangun *broker cluster* menggunakan *message broker ActiveMQ*. Penelitian ini diharapkan dapat meminimalisir kendala pada komunikasi antara *publisher* dan *subscriber* ketika *broker* tidak dapat tersedia. Penelitian ini juga diharapkan dapat meningkatkan tingkat *availability* dari *broker* yang menjadi media komunikasi dalam protokol MQTT.

2. ANALISIS KEBUTUHAN

Tahapan analisis kebutuhan sistem menjadi dasar untuk perancangan dan implementasi sistem. Perancangan dan implementasi secara sistematis memenuhi kebutuhan yang diperlukan oleh sistem agar penelitian sesuai dengan tujuan yang ingin dicapai. Kebutuhan sistem pada penelitian ini dijabarkan sebagai berikut:

a. MQTT *publisher* dapat mengirimkan pesan berbasis topik dalam bentuk *string* kepada MQTT *broker*.
b. MQTT *broker* dapat menerima pesan yang dikirimkan oleh MQTT *publisher*.
c. MQTT *subscriber* dapat melakukan *subscribe* pada topik yang diinginkan kepada MQTT *broker*.
d. MQTT *broker* dapat menyampaikan pesan dari MQTT *publisher* kepada MQTT *subscriber*.
e. MQTT *broker* dapat membentuk *cluster* yang terdiri dari *master* dan *slave broker*, serta dapat mengimplementasikan metode *failover* antara kedua *broker* dalam *cluster*.
f. MQTT *broker* melalui *master broker* dapat menjembatani komunikasi MQTT *publisher* dan *subscriber*. Ketika *master broker* tidak aktif, *slave broker* dapat menangani komunikasi MQTT *publisher* dan *subscriber*.

3. PERANCANGAN



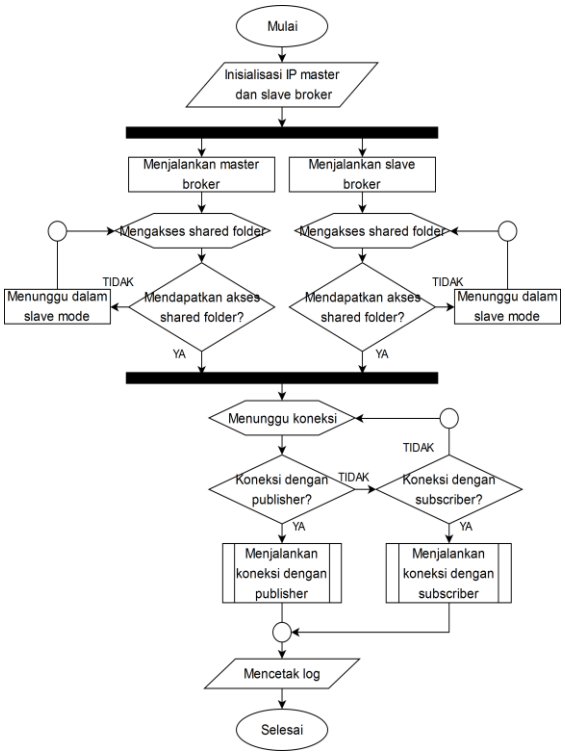
Gambar 1. Perancangan sistem

Berdasarkan Gambar 1, perancangan sistem dilakukan dengan merancang 3 komponen utama yaitu MQTT *publisher*, *subscriber* dan *broker*. Perancangan MQTT *publisher* dilakukan dengan merancang perangkat lunak yang dapat membuat koneksi dengan salah satu MQTT *broker* yang berada dalam *broker cluster* dan melakukan *publish* pesan berbasis topik. Perancangan MQTT *subscriber* dilakukan untuk merancang perangkat lunak yang digunakan untuk proses komunikasi dengan *master* ataupun *slave broker* yang berada dalam *cluster* sehingga dapat melakukan *subscribe* pesan dan menerima pesan dari *broker*. Perangkat lunak MQTT *publisher* dan *subscriber* juga dirancang agar dapat memiliki mekanisme berpindah dari *master* kepada *slave broker* ketika terjadi kegagalan. Perancangan *broker* bertujuan untuk dapat mengimplementasikan *failover* dengan membangun *cluster* yang terdiri dari *master broker*, *slave broker* dan NFS *server*. *Master broker* dirancang agar dapat menjadi MQTT *broker* utama dalam *cluster* dan memiliki cadangan *slave broker* yang berada dalam kondisi pasif menunggu *master broker* mengalami kegagalan. Komunikasi kedua broker dilakukan pada *Sharing File System* yang menggunakan mekanisme komunikasi dari *Network File System* (NFS).

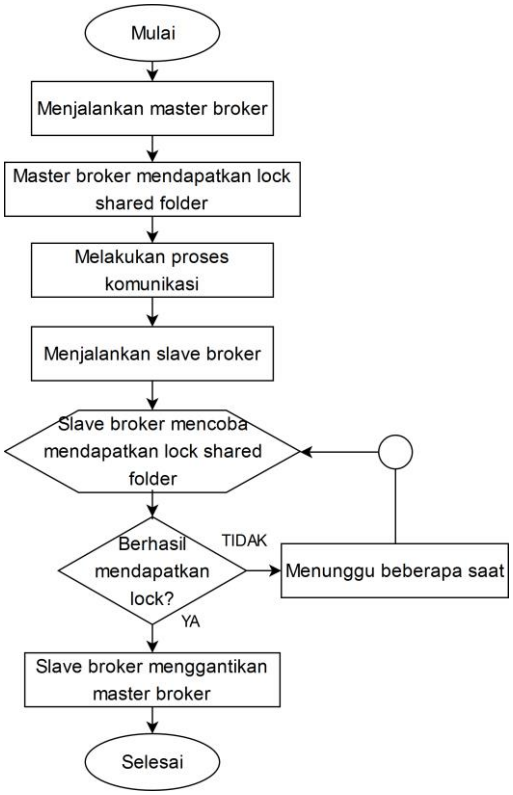
3.1 Perancangan MQTT *broker*

Gambar 2 menjelaskan proses yang dilakukan oleh *broker cluster master* dan slave *broker* dalam sistem. Proses dimulai dari menjalankan layanan milik *master* dan *slave broker* pada Raspberry. *Master* dan slave *broker* melakukan perulangan untuk mengakses *Sharing File System* yang telah ditentukan direktorinya. MQTT *broker* yang mendapatkan akses *Sharing File System* akan menjadi *master broker* dan siap menerima koneksi. Apabila

tidak berhasil mendapatkan akses dari *Sharing File System*, maka MQTT *broker* harus menunggu dalam *slave mode* dan melakukan perulangan untuk mendapatkan akses dari *Sharing File System*. *Master broker* yang sudah aktif dapat menerima koneksi dari MQTT *publisher* atau *subscriber*. Saat ada *request* koneksi dari MQTT *publisher*, maka MQTT *broker* melakukan sub proses komunikasi dengan *publisher*. Saat ada request koneksi dari *subscriber*, maka MQTT *broker* menjalankan sub proses komunikasi MQTT *subscriber*.



Gambar 2. Diagram alir MQTT *broker*



Gambar 3. Diagran alir *failover*

Gambar 3 menunjukkan diagram alir ketika mengimplementasikan *failover*. Proses *failover* diawali dengan *master broker* yang terlebih dahulu dijalankan agar *master broker* mendapatkan *lock* direktori *Sharing File System*. Selanjutnya, *master broker* siap menerima koneksi dari MQTT *publisher* dan *subscriber*. Kemudian, *slave broker* mulai dijalankan dan memasukki *slave mode* untuk menunggu *master broker* tidak aktif. Untuk mendeteksi kegagalan pada *master broker*, *slave broker* mengirimkan *packet* kepada *master broker* secara berulang-ulang. *Slave broker* mengirimkan *packet* dengan protokol NFS versi 4 untuk mencoba mendapatkan *lock* dari *Sharing File System*. *Slave broker* mengirimkan *packet Call* kepada *slave broker* dalam interval 10 detik. *Slave broker* melakukan komunikasi dalam 3 tahapan yaitu *OPEN* untuk membuka komunikasi, *LOCK* untuk mengakses *lock* dari *Sharing File System*, dan *CLOSE* untuk menutup komunikasi bila terjadi kegagalan mendapatkan *lock*. Pada tahapan mengakses *LOCK*, *slave broker* akan mendapatkan *reply* NFS4ERR_DENIED dari *master broker* apabila *master broker* masih dalam keadaan *up* dan belum melepas *lock*. Ketika *master broker* telah terhenti, *slave broker* tidak mendapatkan *reply* NFS4ERR_DENIED melainkan hanya berupa *LOCK* seperti *reply*

master broker ketika mengkonfirmasi *OPEN* dan *CLOSE*. Hal itu menunjukkan *master broker* telah terhenti dan melepas *lock* pada direktori *Sharing File System*. Selanjutnya *slave broker* mengirimkan *packet GETATTR* untuk mendapatkan data dari direktori *Sharing File System* yang ada pada *master broker*. *Slave broker* menggantikan *master broker* ketika pengambilan data pada *master broker* telah selesai.

4. HASIL DAN PEMBAHASAN

4.1 Pengujian *Availability*

Tabel 1. Pengujian *availability*

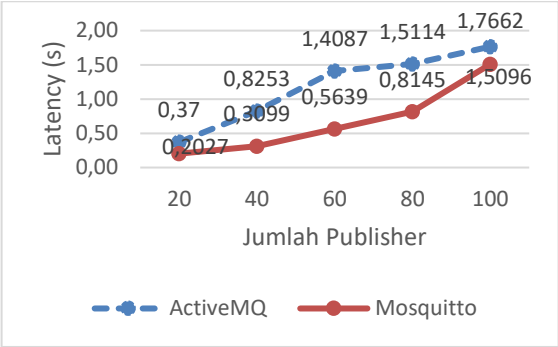
Pengujian Ke -	*Downtime* (s)
1	24.086
2	24.102
3	20.936
4	26.442
5	26.067
Rata-Rata	24.3266

Pengujian dilakukan dengan metode *planned downtime* ketika MQTT *publisher* mengirimkan pesan kepada MQTT *subscriber* secara berulang-ulang. *Master broker* akan dinonaktifkan ditengah-tengah proses komunikasi. Pengujian melihat nilai *downtime* dalam detik ketika *master broker* berpindah kepada *slave broker*.

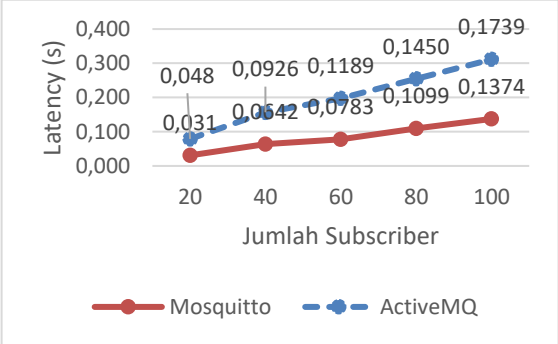
Tabel 1 menunjukkan nilai rata-rata *downtime* adalah 24.3266 detik dari 5 percobaan. Nilai *downtime* yang didapatkan merupakan waktu yang dibutuhkan *slave broker* untuk mendeteksi kegagalan pada *master broker* dan menjalankan *ActiveMQ*. *Slave broker* mendeteksi kegagagalan atau *failure detection* dipengaruhi oleh *cost* dalam hal *processing delay* dari sistem. Karena untuk mendeteksi kegagalan pada *master broker*, *slave broker* mengirimkan *packet* NFSV4 *Call* kepada *master broker* untuk mendapatkan *lock* pada *sharing file system*. Sementara lama waktu yang dibutuhkan *slave broker* menjalankan *ActiveMQ* dipengaruhi nilai *overhead* dalam hal penggunaan CPU dan *memory* ketika mengimplementasikan sistem. Hal itu disebabkan karena *slave broker* membutuhkan waktu yang lebih banyak untuk menjalankan perangkat lunak *ActiveMQ* yang menggunakan CPU dan *memory* yang besar.

4.2 Pengujian *Latency*

Pengujian dilakukan dengan menggunakan *thread* sebanyak 20, 40, 60, 80 dan 100 ketika melakukan komunikasi antara MQTT *publisher* menuju MQTT *subscriber*.



Gambar 4. Pengujian *latency* MQTT *publisher*



Gambar 5. Pengujian *latency* MQTT *subscriber*

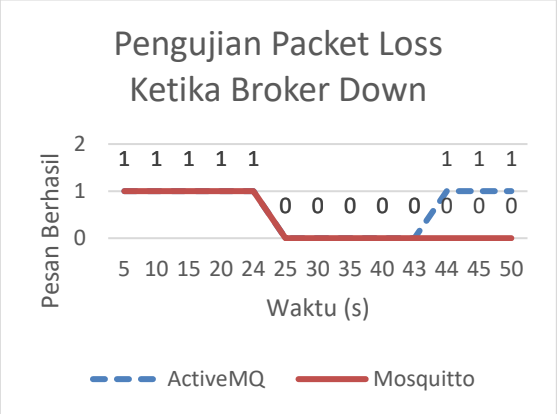
Berdasarkan Gambar 4 dan 5, jumlah *thread* mempengaruhi nilai *latency* yang didapatkan. Semakin banyak penggunaan *thread* mengakibatkan semakin banyak koneksi untuk melakukan *publish* atau *subscribe* sehingga menyebabkan jaringan semakin terbebani. Hasil yang didapatkan menunjukkan semakin banyak jumlah *thread* menyebabkan peningkatan nilai *latency*. Hasil *ActiveMQ* didapatkan lebih besar dibandingkan *Mosquitto*. Hal itu disebabkan karena perangkat keras *broker* yaitu Raspberry Pi 3 Model B memiliki *resource* yang terbatas dan *ActiveMQ* menggunakan *resource* yang besar dalam hal penggunaan CPU dan *memory*. Ketidaksesuaian *overhead ActiveMQ* dengan kemampuan perangkat keras menyebabkan terjadinya penurunan kecepatan ketika melakukan proses komunikasi.

4.3 Pengujian *Packet Loss*

Pengujian dilakukan dengan mengirimkan pesan dari MQTT *publisher* kepada *subscriber* setiap detiknya selama 50 detik. Ditengah proses komunikasi, MQTT *broker* akan dimatikan.

Pengujian melihat pesan yang berhasil diterima pada *ActiveMQ* dan *Mosquitto* setiap detiknya ketika berkomunikasi dan terjadi *down* pada MQTT *broker*.

Gambar 6 menunjukkan jumlah pesan yang berhasil dari detik 1 hingga 50 pada *ActiveMQ* dan *Mosquitto*. Ketika MQTT *broker* dihentikan pada detik 24, Pengiriman pesan pada *Mosquitto* terhenti karena kehilangan koneksi dengan MQTT *broker* sehingga jumlah pesan berhasil pada *Mosquito* mulai detik 25 hingga detik 50 terhenti. Sementara *ActiveMQ* menggunakan *failover* dan berpindah kepada *slave broker* ketika MQTT *broker* terhenti sehingga proses pengiriman pesan pada *ActiveMQ* terjadi packet loss pada detik 25 hingga *slave broker* tersedia pada detik 44. Proses *publish* dilanjutkan ketika *slave broker* telah menggantikan *master broker*. Sementara pada sistem satu *broker* seperti *Mosquitto*, proses komunikasi tidak dapat dilanjutkan ketika *broker* terhenti. MQTT *Publisher* dan *subscriber* harus menunggu hingga MQTT *broker Mosquitto* diaktifkan kembali dalam waktu yang tidak pasti sehingga terjadi packet loss dalam jumlah yang banyak. Oleh karena itu, *ActiveMQ* memiliki jumlah packet loss yang lebih rendah bila dibandingkan dengan *Mosquitto*.



Gambar 6. Pengujian *packet loss*

5. KESIMPULAN

Kesimpulan dari penelitian dijelaskan sebagai berikut:

1. Implementasikan metode *failover* pada MQTT *broker* dapat dilakukan dengan menggunakan *broker cluster* yang terdiri dari *master broker* serta *slave broker*. *Broker cluster* diimplementasikan dengan adanya fitur dari *message broker ActiveMQ*. *Master* dan *slave broker* menggunakan *Sharing File System, KahaDB* serta *Network File System* (NFS) agar kedua MQTT *broker* saling terhubung dan membentuk *broker cluster*.
2. Implementasi metode *failover* dengan *ActiveMQ* dapat memperbaiki *availability* dari MQTT *broker* dengan adanya *Sharing File System* antara *master* dan *slave broker*. Kesimpulan hasil analisis dijelaskan sebagai berikut:
   - Nilai rata-rata *downtime* yang didapatkan sebesar 24.3266 detik dari lima percobaan. Waktu tersebut adalah waktu yang dibutuhkan untuk *slave broker* mendeteksi *master broker* telah terhenti dan menjalankan *ActiveMQ*. *Downtime* yang terjadi dipengaruhi *cost* dalam hal *processing delay* dan *overhead ActiveMQ* terkait penggunaan CPU dan *memory*.
   - Nilai *latency* sistem memiliki hasil lebih besar bila dibandingkan dengan sistem MQTT pada umumnya yang menggunakan *Mosquitto*. Hasil pengujian latency sistem yang didapatkan adalah 0.37 s, 0.8253 s, 1.4087 s, 1.5114 s, 1.7662 s pada sisi MQTT *publisher* dan 0.048 s, 0.0926 s, 0.1189 s, 0.1450 s, 0.1739 s pada sisi MQTT *subscriber*. Hasil pengujian *latency* dipengaruhi faktor penggunaan *thread* yaitu 20, 40, 60, 80 dan 100. Hasil juga dipengaruhi oleh *overhead* penggunaan CPU dan *memory* yang lebih besar karena mengakibatkan penurunan kecepatan ketika melakukan komunikasi.
   - Nilai *packet loss* ketika terjadi kegagalan pada MQTT *broker* lebih sedikit bila dibandingkan dengan *Mosquitto*. *ActiveMQ* memiliki hasil 20 *packet loss* dalam komunikasi 50 detik dan terjadi kegagalan pada MQTT *broker*. *Packet loss* yang terjadi menggambarkan waktu *downtime* sistem ketika terjadi kegagalan. Sistem dapat meminimalisir terjadinya *packet loss* dengan adanya *slave broker* yang meneruskan proses komunikasi ketika terjadi kegagalan pada *master broker*. Sementara pada sistem dengan broker tunggal seperti *Mosquitto*, proses komunikasi harus terhenti karena harus menunggu *broker* diperbaiki sehingga terjadi *packet loss* yang besar.

6. DAFTAR PUSTAKA

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. 2015. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communication Surveys & Tutorials*, Vol. 17, No. 4, Fourth Quarter 2015, p. 2347 – 2376.

Happ, D. & Wolisz, A. 2016. Limitations of the Pub/Sub Pattern for Cloud Based IoT and Their Implications. Telecommunication Networks Group (TKN), Technische Universitat Berlin.

Harsapranata, A. I. 2015. Implementasi Failover Menggunakan Jaringan VPN dan Metronet Pada Astridogroup Indonesia. *Jurnal Teknik dan Ilmu Komputer*, Vol. 04, No. 13, Jan – Mar 2015, p. 69 – 77.

Hayun, D. R. L., & Wibisono, W. 2017. Optimasi Pemilihan Child Broker Pada Model Komunikasi Publish/Subscribe Pada Protokol Data Distribution Service di Area Multi-Zone. *JUTI*, Vol. 15, No. 01, Jan 2017, p. 11 - 25.

Hunkeler, U., Truong, H. L., & Stanford-Clark, A. 2008. MQTT-S - A Publish/Subscribe Protocol For Wireless Sensor Networks. *Communication Systems Software and Middleware and Workshops (COMWARE)* 2008.

Ionescu, V. M. 2015. The Analysis of The Performance of RabbitMQ and ActiveMQ. Faculty of Electronics, Communications and Computer Science, University of Pitesti, Romania.

Juliharta, I G. P. K., Supedana, W., & Hostiadi, D. P. 2015. High Availability Web Server Berbasis Open Source Dengan Teknik Failover Clustering. *Seminar Nasional Teknologi Informasi dan Multimedia* 2015, 6 – 8 Feb 2015, p. 31 – 36.

Kahanwal, B., & Singh, T. P. 2012. The Distributed Computing Paradigms: P2P, Grid, Cluster, Cloud, and Jungle. *International Journal Of Latest Research in Science and Technology*, Vol. 1, Issue 2, Jul – Aug 2012, p. 183-187.

Magnoni, L. 2015. Modern Messaging for Distributed Systems. *Journal of Physics: Conference Series 608* (2015) 012038.

Petersen, H., Bacelli, E., & Wahlisch, M. 2014. Interoperable Services on Constrained Devices in the Internet of Things. Freie Universitat Berlin, Germany.

Pribadi, P. T. 2013. Implementasi High-Availability VPN Client Pada Jaringan Komputer Fakultas Hukum Universitas Udayana. *Jurnal Ilmu Komputer*, Vol., 6 No. 1, April 2013, p. 17 - 24.

Tarigan, S. O. F., Sitepu, H. I., & Hutagalung, M. 2014. Pengukuran Kinerja Sistem Publish/Subscibe Menggunakan Protokol MQTT (Message Queueing Telemetry Transport). *Jurnal Telematika*, Vol. 9, No. 1. Institut Teknologi Harapan Bangsa, Bandung.

Thangavel, D., Ma, X., Valera, A., Tan, H., & Tan, C. K. 2014. Performance Evaluation of MQTT and CoAP via a Common Middleware. 2014 IEEE 9th *Internasional Conference On Intelligent Sensors, Sensor Networks And Information Processing (ISSNIP) Symposium On Sensor Networks*, 21-24 April 2014, Singapore.

Yokotani, T., & Sasaki, Y. 2016. Comparison with HTTP and MQTT on Required Network Resources for IoT. *The 2016 Interntional Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC*).