Analisis Penggunaan Metode *3 in 1 Heartbeat* Pada Arsitektur *Active-Active Distributed Controller* di Jaringan *Software Defined Networks*

Mutiara Ramadhani Wijaya[1], Vera Suryani[2], Muhammad Arief Nugroho[3]

[1,2,3]Fakultas Informatika, Universitas Telkom, Bandung
[1]mutiarawijayaa@students.telkomuniversity.ac.id, [2]verasuryani@telkomuniversity.ac.id,
[3]arif.nugroho@telkomuniversity.ac.id

Abstrak

Masalah yang terdapat pada *Distributed Controller (active-active)* dengan penggunaan *synchronous message exchange* yaitu setiap pesan yang dikirimkan oleh *controller* A (*sender*) akan direspon oleh *controller* B (*receiver*), kemudian dapat melakukan proses pengiriman pesan berikutnya setelah menerima respon tersebut. Hal ini dapat menyebabkan penurunan kinerja serta menambah beban kerja pada *controller* karena setiap pesan yang akan direspon membutuhkan proses secara langsung untuk menghasilkan *acknowledgment.* Oleh karena itu, dalam penelitian ini dilakukan pengembangan untuk meningkatkan mekanisme *message exchange* dan mengurangi beban sumber daya pesan dalam melakukan pengiriman informasi pesan antara *controller*. Penelitian ini mengusulkan untuk menggunakan *asynchronous message* sebagai *message exchange* dengan metode *3 in 1*, yaitu mekanisme yang dilakukan dengan proses pengiriman tiga pesan dan menghasilkan *reply* dalam satu *acknowledgment.* Hasil pengujian kedua metode tersebut memperoleh nilai presentase tertinggi CPU *Usage* pada metode *3 in 1* sebesar 7,90% sedangkan nilai tertinggi pada metode satu pesan satu *acknowledgment* sebesar 9,57%. Berdasarkan hasil tersebut metode *3 in 1* memperoleh nilai selisih 1,67% lebih rendah dibandingkan dengan metode satu pesan satu *acknowledgment.* Oleh karena itu, dengan penggunaan metode *3 in 1* beban kerja pada masing-masing *controller* berkurang. Metode *3 in 1* juga membuktikan hasil kinerja yang lebih tinggi serta waktu *failover* yang lebih cepat.

Kata kunci : *asynchronous, distributed controller, message exchange, software defined network*

Abstract

The problem of Distributed Controllers (active-active) using synchronous message exchange is that every message sent by controller A (sender) will be responded by controller B (receiver), then can make the process of sending the next message after receiving that response. This can cause a decrease in performance and increase the workload on the controller because every message that will be responded to requires a direct process to produce acknowledgments. Therefore, this research was developed to improve the message exchange mechanism and reduce the burden of message resources in sending message information between controllers. This study proposes to use asynchronous messages as message exchanges with the 3 in 1 method, which is the mechanism carried out by the process of sending three messages and producing a reply in one acknowledgment. The test results of the two methods obtained the highest percentage of CPU Usage on the 3 in 1 method of 7.90% while the highest value on the one-message one-acknowledgment method was 9.57%. Based on these results the 3 in 1 method obtained a difference value of 1.67% lower than the one-message one-acknowledgment method. Therefore, with the use of the 3 in 1 method the workload on each controller is reduced. The 3 in 1 method also proves higher performance results and faster failover times.

Keywords: *asynchronous, distributed controller, message exchange, software defined network*

1. Pendahuluan

Latar Belakang

Distributed Controller merupakan *multiple-controller* yang digunakan untuk meningkatkan skalabilitas dalam jaringan. Skalabilitas merupakan salah satu yang harus diperhatikan dalam suatu sistem, jaringan dan proses yang menunjukkan kemampuannya untuk menangani jumlah peningkatan kinerja yang berpotensi besar. *Distributed Controller* dibagi menjadi dua arsitektur yaitu *flat architecture* dan *hierarchical architecture*. *Flat architecture* merupakan *controller* yang diposisikan secara horizontal sehingga *control plane* hanya terdiri dari satu *layer* dan masing-masing *controller* memiliki tanggung jawab yang sama pada waktu yang sama [1]. *Flat architecture* memberikan lebih banyak ketahanan terhadap kegagalan dalam jaringan dibandingkan dengan *hierarchical architecture* karena memiliki beberapa *layer* dengan tanggung jawab yang berbeda sehingga lebih rentan mengalami kegagalan terhadap jaringan [2]. *Flat architecture Distributed Controller* dibagi menjadi dua yaitu, *active-active* dan *active-passive*. *Distributed Controller (active-active)* dengan menggunakan *flat architecture* membuat semua *controller* aktif dan bekerja sama untuk mengelola suatu jaringan secara bersamaan serta dapat memperluas kapabilitas dari control plane itu sendiri.

Pada penelitian sebelumnya dilakukan penelitian *Distributed Controller (active-active)* yang menggunakan *synchronous message exchange*. Dibandingkan dengan metode *active-backup* penelitian tersebut menghasilkan

presentase kinerja pada CPU lebih rendah mengakibatkan proses failover menjadi lebih cepat [3]. Secara umum *synchronous message* memiliki kekurangan yaitu membutuhkan waktu yang lebih banyak ketika menunggu respon dari *receiver* karena komunikasi yang bersifat *real-time* dan beban kerja pada setiap *controller* lebih besar. Berbeda *dengan asynchronous message*, beberapa pesan dapat dikirimkan secara langsung kemudian *receiver* akan mengeksekusi. Penelitian ini merupakan pengembangan dari penelitian sebelumnya yaitu *asynchronous message*. Proses yang akan dilakukan yaitu menggunakan metode *3 in 1* yang bekerja dengan cara mengirimkan tiga pesan kemudian akan direspon oleh *receiver* dengan satu *acknowledgment*. Dengan menggunakan metode tersebut dapat mengurangi beban kerja masing-masing *controller* dan proses pengiriman pesan lebih cepat.

Topik dan Batasannya

Dalam penelitian sebelumnya menggunakan *synchronous message* sebagai *message exchange* pada *Distributed Controller (active-active).* Proses ini dilakukan secara *real-time* yaitu dengan mengirim satu pesan kemudian akan segera dilakukan *reply* dan begitupun seterusnya. Hal ini mengakibatkan penurunan kinerja, beban kerja terhadap *controller* lebih besar serta membutuhkan waktu yang lebih banyak. Oleh karena itu, penelitian ini mengusulkan untuk mengembangkan mekanisme *message exchange* tersebut dengan *asynchronous message* menggunakan metode *3 in 1*. Metode ini dilakukan dengan cara mengirimkan tiga pesan kemudian di *reply* dengan satu *acknowledgment* begitupun seterusnya. Dengan menggunakan metode ini dapat mengurangi beban kerja pada setiap controller serta waktu yang dibutuhkan lebih cepat.

Pada penelitiain ini melakukan pengujian menggunakan parameter CPU *Usage* untuk menganalisa kinerja beban kerja (*workload*) terhadap controller serta menggunakan parameter dengan mekanisme *failover time* untuk melihat waktu *failover* yang terjadi ketika melakukan *switch migration* terhadap *controller*. Kemudian paramater *throughput* untuk membandingkan kinerja pada kedua metode *message exchange*. *Controller* yang digunakan pada penilitian ini adalah *POX controller.* Kemudian, dalam melakukan simulasi pada penelitian ini menggunakan tiga *virtualbox* yang digunakan sebagai *controller* 1, *controller* 2, dan juga mininet.

Tujuan

Tujuan dilakukannya penelitian ini adalah menganalisis kinerja *asynchronous message exchange* pada *distributed controller (active-active)* dengan menggunakan metode *3 in 1* terhadap beban kerja serta waktu dalam proses *message exchange* dan membandingkannya dengan komunikasi *synchronous message*. Parameter yang digunakan dalam penelitian ini meliputi nilai CPU *usage* dan *failover* time untuk menganalisa beban kerja terhadap *controller* serta waktu *failover* yang terjadi. Kemudian, parameter *throughput* untuk melakukan perbandingan kinerja metode *3 in 1* dan satu pesan satu *acknowledgment* (*synchronous message*).

Organisasi Tulisan

Bagian selanjutnya terdiri atas studi terkait, sistem yang di bangun, evaluasi, dan kesimpulan. Pada bagian studi terkait akan dijelaskan studi literatur dan teori yang menjadi dasar pengembangan dan pelaksanaan penelitian ini. Penjelasan mengenai rancangan dan implementasi sistem akan dijelaskan lebih lanjut pada bagian sistem yang dibangun. Dari implementasi tesebut akan dilaksanakan beberapa skenario untuk melakukan pengujian terhadap sistem. Hasil dan analisis berdasarkan uraian pada bagian evaluasi juga dituliskan pada bagian kesimpulan.

2. Studi Terkait

2.1. Related Works

Pada jurnal atau paper sebelumnya telah dibahas penelitian terkait dengan penelitian yang akan dilakukan diantaranya seperti pada Tabel 2.1.

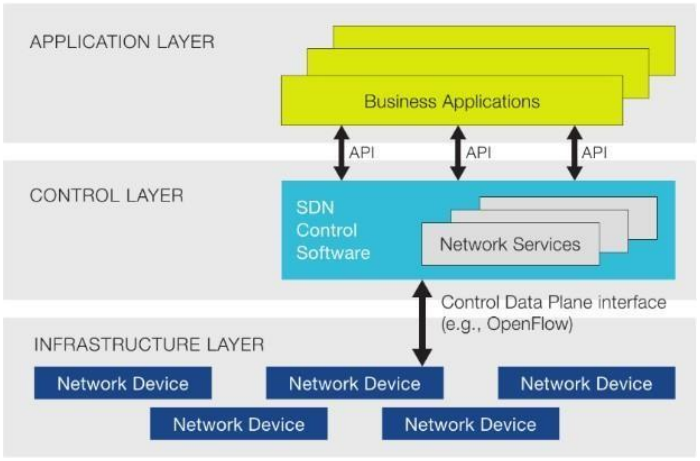
Tabel 2.1. *Related Works*

Judul Paper	Metode	Hasil
An Overview on SDNArchitectures with Multiple Controllers (2016)	*Flat Architecture vs Hirarchical Architecture*	*Flat architecture* memberikan lebih banyak ketahanan terhadap kegagalan dalam jaringan dibandingkan dengan *hierarchical controller* karena memiliki beberapa *layer* dengan tanggung jawab yang berbeda sehingga lebih rentan mengalami kegagalan terhadap jaringan [2].
AR2C2: Actively Replicated Controllers for SDN Resilient Control Plane (2016)	*Active-Passive*	Adanya *Backup Controller* berfungsi untuk memantau dan juga melakukan *backup* jika terjadi kegagalan jaringan pada *controller* utama [4].

Desain *Distributed Controller* dengan Metode *Active-Active* Pada Jaringan *Software Defined Network* (2019)	*Active-Active*	*Flat architecture* menggunakan metode *active-active* membuat semua *controller* aktif bekerja sama untuk mengelola suatu jaringan secara bersamaan dan lebih *frequent* untuk menjamin konsistensi jaringan [3].
New method for controller-to-controller communication in distributed SDN architecture (2017)	*Message exchange asynchronous message and synchronous message*	Penelitian ini melakukan perancangan dan implementasi *interface* komunikasi *controller to controller*. Pada penelitian ini juga menjelaskan jenis komunikasi *synchronous* dan *asynchronous message.* Hasil penilitian menunjukkan bahwa penggunaan *asynchronous message* meningkatkan fleksibilitas jaringan yang secara signifikan meningkatkan kinerja jaringan serta mengurangi beban kerja pada *controller*[13].

2.2. Software Defined Network

Software Defined Network (SDN) merupakan paradigma baru yang dapat diprogram dan dikontrol secara terpusat dengan memisahkan *control plane* dan *data plane*. *Control plane* merupakan komponen pada jaringan yang berfungsi untuk mengontrol jaringan dengan melakukan konfigurasi sistem, manajemen jaringan, menentukan informasi *routing table* dan *forwarding table*. Kemudian, *data plane* merupakan bagian yang bertanggung jawab untuk meneruskan paket (*forwarding*). Arsitektur SDN, memungkinkan suatu jaringan dapat meningkatkan skalabilitas jaringan yang diwujudkan melalui impelementasi sederhana dari penambahan komponen dan layanan jaringan. SDN meperkenalkan suatu metode untuk meningkatkan tingat abstraksi pada konfigurasi jaringan. Tujuan utama dari SDN adalah mencapai pengelolaan jaringan yang lebih baik dengan kompleksitas yang besar serta memastikan bahwa semua keputusan dari sistem kontrol dibuat dari titik pusat (*controller*) [6].



Gambar 2. 1. Arsitektur *Software Defined Network* (SDN)

2.3. Distributed Controller

Distributed Controller adalah *multiple-controller* pada suatu jaringan *Software Defined Network* (SDN) yang digunakan untuk meningkatkan skalabilitas dalam jaringan. Pada *controller* ini, SDN memiliki dua atau lebih *controller* dimana masing-masing *controller* bertanggung jawab untuk memberikan informasi atau pesan kepada *controller* lainnya [7]. *Distributed Controller* dibagi menjadi dua arsitektur *yaitu flat architecture* dan *hierarchical architecture*. Keduanya dapat meningkatkan *switch* ataupun *controller latency*. Namun pada *flat architecture* lebih mempertahankan terhadap *failure* pada jaringan dibandingkan dengan *hierarchical architecture*. *Flat architecure memilki* tanggung jawab yang sama dengan satu waktu, sedangkan *hierarchical architecture* memiliki tanggung jawab yang berbeda karena terbagi dengan *multiple levels* [2].

2.4. OpenFlow Protocol

Jaringan *OpenFlow* adalah arsitektur jaringan yang dikerahkan secara efisien untuk SDN [8]. Protokol *OpenFlow* merupakan protokol yang dirancang dengan tujuan untuk mengontrol data plane (*forwarding*) pada *switch* yang telah dipisahkan dari *control plane* menggunakan *controller* di sebuah *server*. *OpenFlow* menawarkan lebih banyak penerusan data yang fleksibel karena OF *switch* sepenuhnya bertanggung jawab untuk melakukan penerusan pesan (*forward*). *OpenFlow* memungkinkan komunikasi dengan *control plane* untuk mengirim instruksi, menerima request, dan juga bertukar informasi [2][5].

2.5. Reliabilitas

Reliabilitas merupakan kemampuan jaringan yang memiliki ketersediaan layanan jaringan yang tinggi. Kemampuan ini mempertimbangkan lalu lintas jaringan seperti *overload*, *latency*, dan juga *throughput* [9]. Reliabilitas memiliki suatu teknik jaringan yaitu mekanisme *failover*, yang memberikan dua jalur koneksi atau lebih ketika salah satu jalur mati, maka koneksi masih tetap berjalan dengan mengalihkan ke jalur lainnya. Mekanisme ini dirancang untuk segera mungkin bertindak ketika terjadi gangguan [3] [10].

2.6. Skalabilitas

Skalabilitas adalah salah satu yang harus diperhatikan dalam suatu sistem, jaringan dan proses yang menunjukkan kemampuannya untuk mengangani jumlah peningkatan kinerja yang berpotensi besar. Skalabilitas pada SDN yang *overload* pada *data plane* dan *control plane* dapat dikurangi dengan menggunakan CPU, RAM dan fungsionalitas file I/O [11].

2.7. Synchronous dan *Asynchornous Message Exchange*

Sistem pertukaran pesan (*message exchange*) merupakan serangkaian teknik yang memungkinkan objek untuk berkomunikasi dalam mengirim dan menerima pesan melalui jaringan. Sebuah pesan berisi informasi seperti *request, reponse*, ataupun pengiriman yang tak terjawab. Dalam *message exchange* terbagi menjadi dua model komunikasi yaitu, *synchronous message* dan *asynchronous message*. *Synchronous message*, memiliki titik sinkronisasi setiap intreval migrasi ketika komunikasi terjadi. Semua proses harus menunggu dan memblokir sampai semua proses mencapai titik yang sama [12]. Pendekatan ini bekerja secara *real-time* artinya proses komunikasi berjalan secara langsung. Ketika *sender* mengirim pesan, maka *receiver* akan melakukan respon. Hal ini berpengaruh dengan eksekusi pesan berikutnya karena *sender* tidak dapat mengirim pesan sebelum menerima respon dari *receiver* sehingga semakin banyak waktu yang dibutuhkan ketika menunggu respon tersebut. Berbeda dengan *asynchronous message*, metode komunikasi ini terdapat sistem antrian (*message queue*) sehingga tidak memerlukan tanggapan atau respon segera untuk melanjutkan proses berikutnya. Pada *asynchronous message*, beberapa pesan dapat dikirimkan secara langsung, kemudian *receiver* akan mengeksekusi. Pendekatan ini memiliki keuntungan yang banyak dibandingkan dengan *synchronous message*. *Asynchronous message* (*nonblocking i/o*) menunjukkan performansi yang tinggi dalam komunikasi *controller-to-controller* dibandingkan dengan *synchronous message* (*blocking i/o*) [13].

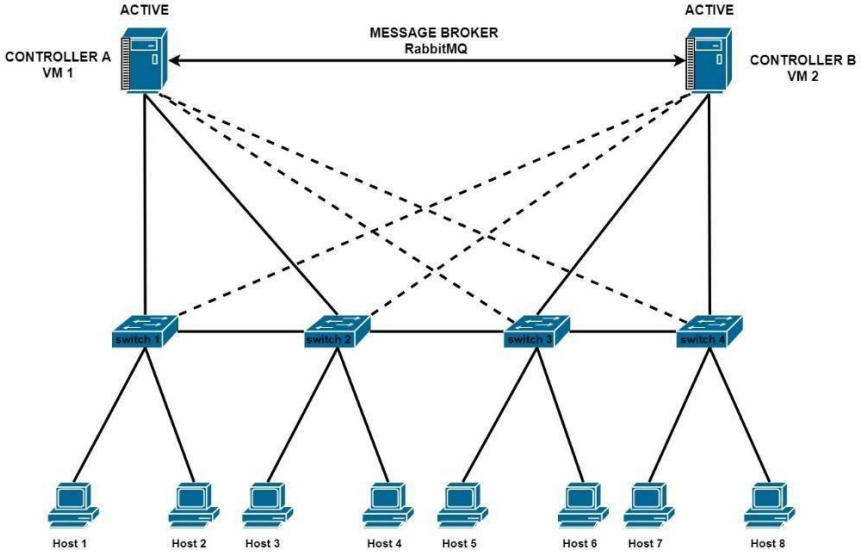
2.8. Metode *3 in 1 Heartbeat*

Metode *3 in 1 Heartbeat* merupakan metode baru yang dilakukan pada penelitian ini dengan menggunakan arsitektur *active-active Distributed Controller.* Proses pertukaran pesan (*message exchange*) akan dilakukan setelah masing-masing *controller* sudah saling mendeteksi bahwa status *controller* sudah saling aktif atau yang disebut dengan *heartbeat*. Kemudian, dilakukan proses pertukaran pesan antar *controller* menggunakan metode *3 in 1*, yang bekerja dengan cara *controller* A sebagai *sender* mengirimkan tiga buah pesan dan *controller* B sebagai *receiver* akan merespon dengan satu *acknowledgment.* Metode *3 in 1* menggagas dari metode yang dilakukan pada *Fast Recovery & Fast Retransmit* dalam *TCP Reno*, dimana metode tersebut melakukan pengiriman tiga buah duplikat pesan untuk mempercepat waktu pengiriman pesan yang hilang. Kemudian terdapat penelitian yang menjelaskan bahwa pada pengiriman duplikat ketiga nilai *threshold* menurun yang mempengaruhi kecepatan pengiriman data terus meningkat secara linier[14]. Berdasarkan hal tersebut, penelitian ini menggunakan metode *3 in 1* pada arsitektur *active-active Distributed Controller* dengan cara mengirimkan tiga buah pesan dan direspon satu *acknowledgment* pada setiap proses *message exchange* antar *controller* dengan tujuan untuk mengurangi beban sumber daya pesan dalam melakukan proses pertukaran pesan sehingga berjalan lebih cepat.

3. Sistem yang Dibangun

3.1. Arsitektur Sistem

Perancangan sistem pada penelitian ini merupakan metode pengembangan dari penelitian sebelumnya yaitu rancangan topologi *Distributed Controller* (*active-active*) [3]. Model komunikasi sebagai *message exchange* pada penelitian yang akan dilakukan menggunakan *asynchronous message* dengan metode *3 in 1*. *Controller* yang digunakan pada penelitian ini adalah *POX controller*.



Gambar 3. 1. Topologi *Distributed Controller (Active-Active)*

Pada Gambar 3.1. terdapat dua buah *controller* yang saling aktif untuk bekerja sama yaitu *Controller* A dan *Controller* B. Masing-masing *controller* mengelola dua buah switch, dimana *switch* 1 dan *swicth* 2 dikelola oleh *controller* A, namun juga terhubung oleh *controller* B sebagai *backup link* yang aktif. Begitupun *switch* 3 dan *switch* 4 yang dikelola oleh *controller* B, namun juga terhubung oleh *controller* A. Ketika salah satu *controller down*, *switch* akan berpindah ke *controller* yang lainnya. Dalam topologi diatas, menggunakan *network* yang sama dan setiap *host* dapat saling berkomunikasi dengan *host* lainnya.

Tabel 3.1. *IP Address*

Devices	**IP Address**
Controller A	192.168.56.103
Controller B	192.168.56.102
Mininet	192.168.56.101
Host 1	192.168.56.1
Host 2	192.168.56.2
Host 3	192.168.56.3
Host 4	192.168.56.4
Host 5	192.168.56.5
Host 6	192.168.56.6
Host 7	192.168.56.7
Host 8	192.168.56.8

Pada Tabel 3.1. menjelaskan *IP Address* pada *controller* dan juga *host* yang digunakan dalam penelitian ini. *IP Address* pada tabel tersebut merepresantasikan dengan topologi pada Gambar 3.1..

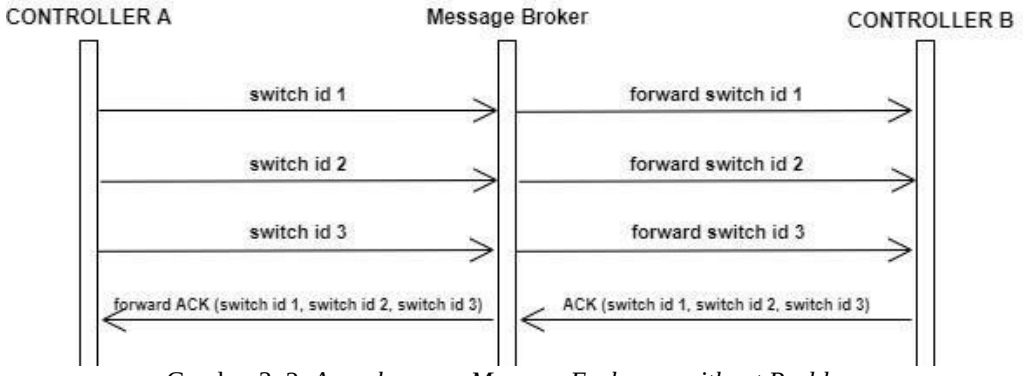
Tabel 3.2. Jumlah *Node*

Switch	4	8	16	32
Host	8	16	32	64

Pada Tabel 3.2. merupakan jumlah *node* yang berisi jumlah *switch* dan jumlah *host* yang akan dilakukan untuk pengujian pada penelitian ini.

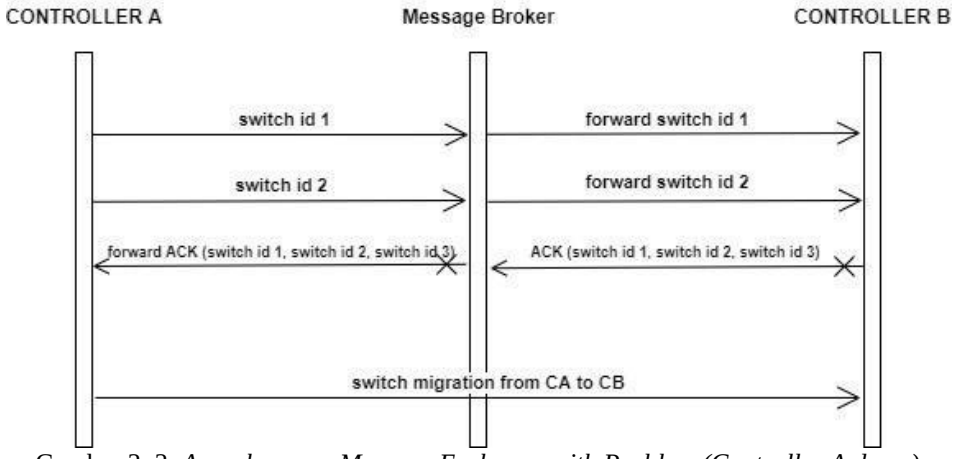
3.2. Arsitektur *Message Exchange*

Pada penelitian ini pengujian proses *message exchange* akan dilakukan dengan dua skenario. Skenario pertama akan dilakukan dengan menjalankan sistem tanpa adanya masalah terhadap *controller*, skenario kedua akan dilakukan dengan cara mematikan controller A sehingga terjadi *down*. Pada penelitian ini proses *message exchange* dilakukan dengan mengirimkan tiga pesan kemudian di respon oleh *controller* B dengan satu *acknowledgment* (ACK). Kemudian, proses pengiriman pesan dilakukan maksimal selama 1 detik, sehingga waktu proses pengiriman pesan yang dilakukan berlangsung selama kurang dari 1 detik. Jika waktu pengiriman lebih dari 1 detik dan pengiriman pesan dilakukan kurang dari tiga pesan, maka *controller* dinyatakan terjadi *down* sehingga dilakukan *switch migration* dari *controller* A terhadap *controller* B. Waktu 1 detik diambil sebagai batasan maksimum waktu pengiriman pesan karena merupakan angka yang cukup optimal sehingga proses pengiriman pesan tidak melebihi waktu yang dapat menyebabkan proses pertukaran pesan menjadi lebih lambat.



Gambar 3. 2. *Asynchronous Message Exchange without Problem*

Pada Gambar 3.2. merupakan *asynchronous message exchange* antar *controller* menggunakan *message broker* (*RabbitMQ*). Pada gambar tersebut *controller* A mengirim pesan melalui *message broker*. Kemudian *message broker* meneruskan pesan tersebut kepada *controller* B. Begitupun sebaliknya, controller B menyampaikan pesan melalui message broker kemudian meneruskan (*forward*) pesan tersebut dengan satu *acknowledgment* (ACK) yang berisi balasan tiga pesan ke *controller* A. Adapun parameter yang digunakan dalam pengujian dengan skenario ini adalah CPU *Usage* untuk melilhat beban kerja *controller*.



Gambar 3. 3. *Asynchronous Message Exchange with Problem (Controller A down)*

Pada Gambar 3.3. merupakan *asynchronous message exchange* yang memiliki masalah *controller* A *down*. Hal ini disebabkan karena pesan yang dikirimkan kurang dari tiga pesan dan juga waktu pengirimannya lebih dari 1 detik. Sehingga dilakukan proses *switch migration* dari *controller* A terhadap *controller* B. Pada skenario ini akan dilakukan pengujian dengan menggunakan mekanisme *failover*.

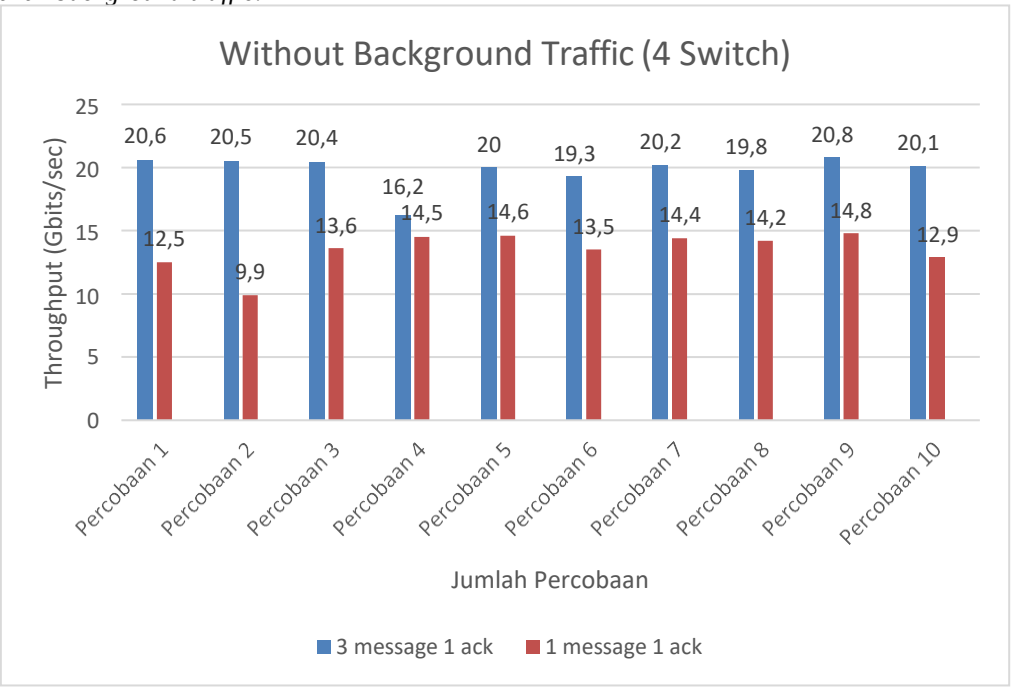
4. Evaluasi

Pada bab ini, menjelaskan terkait dengan hasil pengujian simulasi yang dilakukan. Pengujian simulasi ini dilakukan berdasarkan dua skenario yaitu skenario sebelum terjadinya *down* dan skenario ketika terjadinya *down*. Parameter yang dilakukan adalah *failover time*, CPU *Usage*, dan *throughput* untuk melihat perbandingan kinerja dari kedua metode yaitu *3 in 1* dan satu pesan satu *acknowledgment*.

4.1. Pengujian Perbandingan Kinerja

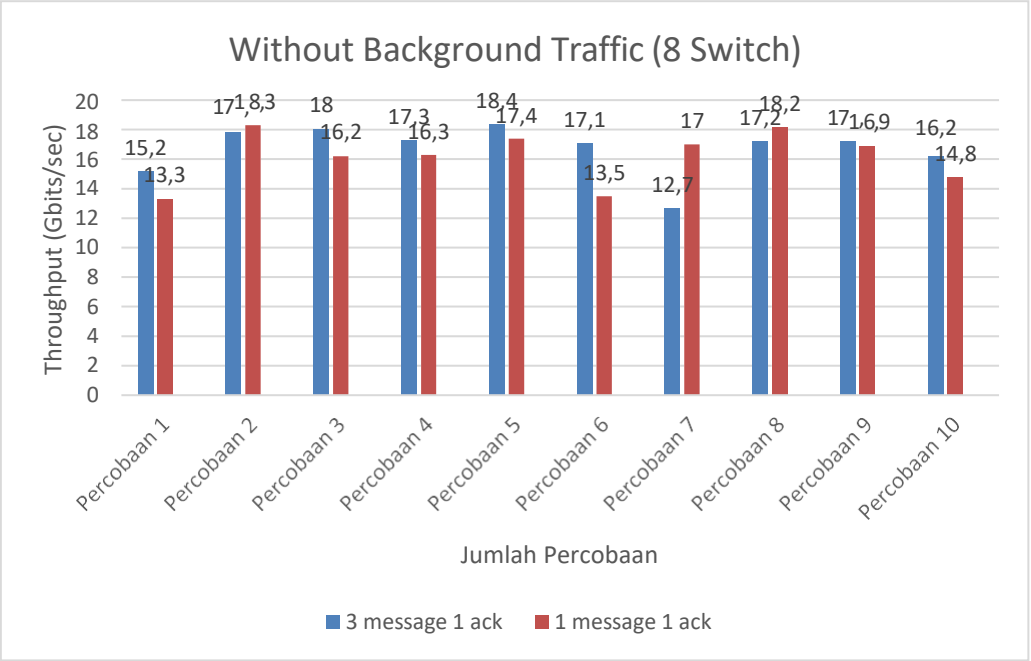
Pengujian ini bertujuan untuk mengetahui perbandingan kinerja metode *3 in 1* dengan satu pesan satu *acknowledgment*. Parameter yang digunakan dalam pengujian ini adalah *throughput*. Pengujian ini dilakukan berdasarkan dua pengujian yaitu uji throughput tanpa menggunakan *background traffic* dan pengujian kedua menggunakan *background traffic*.

Pengujian *throughput* tanpa menggunakan *background traffic* dilakukan dengan cara memanggil perintah *iperf* pada terminal *xterm* host 1 dan host 8 untuk 4 *switch*, host 1 dan host 16 untuk 8 *siwtch*, host 1 dan host 32 untuk 16 *switch*, serta host 1 dan host 64 untuk 32 *switch*. Pengujian dilakukan sebanyak 10 kali percobaan dan diuji selama 10 detik pada masing-masing topologi. Berikut adalah grafik hasil pengujian nilai *throughput* tanpa menggunakan *background traffic*.



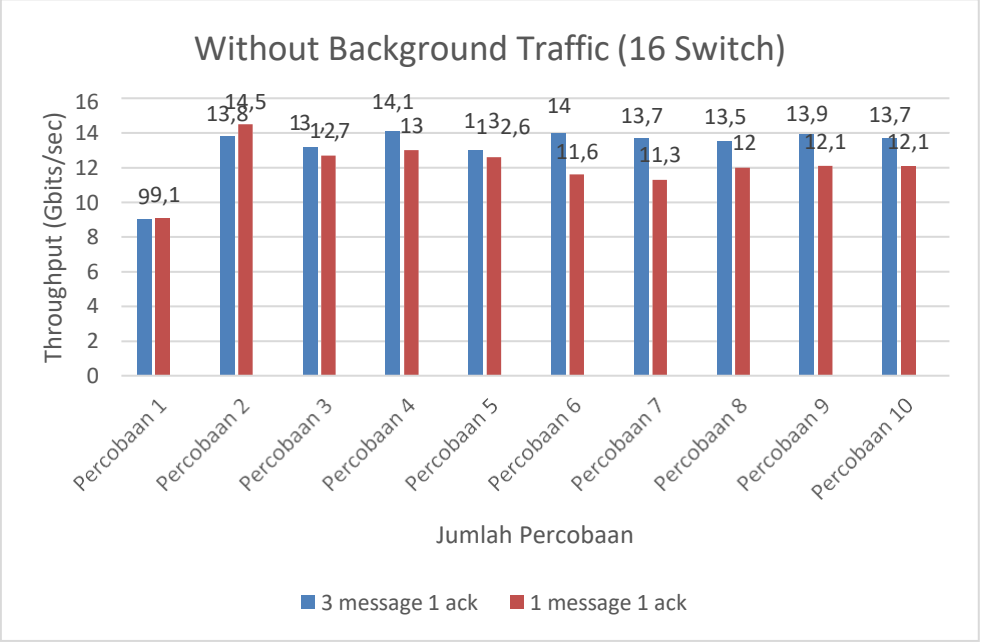
Gambar 4. 1. Pengujian *throughput* tanpa background traffic sebanyak 4 *switch*

Berdasarkan hasil 10 percobaan yang dijelaskan dalam grafik gambar 4.1., masing-masing dilakukan selama 10 detik. Didapatkan hasil berupa nilai *throughput* pada metode *3 in 1* dengan 4 *switch* berkisar 16,2-20,8 Gbits/sec dan nilai rata-rata sebesar 19,79 Gbits/sec. Sedangkan nilai *throughput* satu pesan satu *acknowledgment* pada 4 *switch* berkisar pada 9,9-14,8 Gbits/sec dengan nilai rata-rata sebesar 13,49 Gbits/sec. Berdasarkan hasil tersebut, nilai *throughput* pada metode *3 in 1* dengan 4 *switch* memperoleh selisih 31,8% lebih tinggi dibandingkan dengan nilai throughput pada metode satu pesan satu acknowledgment.



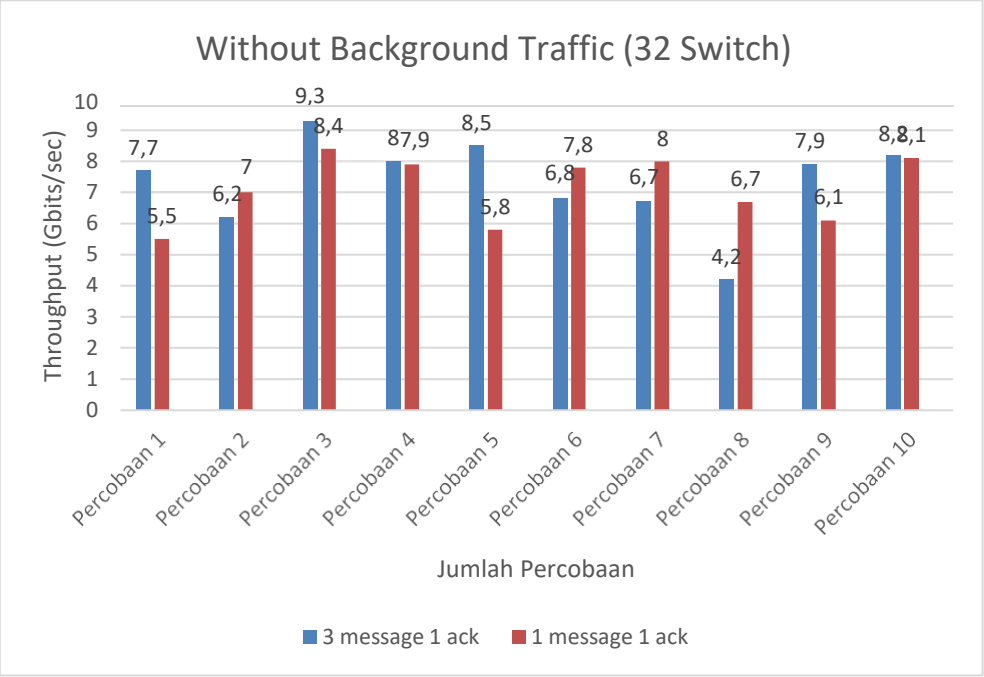
Gambar 4. 2. Pengujian *throughput* tanpa *background traffic* sebanyak 8 *switch*

Berdasarkan grafik pada gambar 4.2., didapatkan hasil berupa nilai *throughput* metode *3 in 1* pada 8 *switch* berkisar 12,7-18,4 Gbits/sec dengan nilai rata-rata sebesar 16,71 Gbits/sec. Sedangkan nilai *throughput* satu pesan satu *acknowledgment* pada 8 *switch* berkisar pada 13,3-18,3 Gbits/sec dengan nilai rata-rata sebesar 16,19 Gbits/sec. Dari hasil tersebut, nilai *throughput* pada metode *3 in 1* dengan 8 *switch* memperoleh selisih 3,1% lebih tinggi dibandingkan dengan nilai *throughput* pada metode satu pesan satu *acknowledgment*.



Gambar 4. 3. Pengujian throughput tanpa background traffic sebanyak 16 *switch*

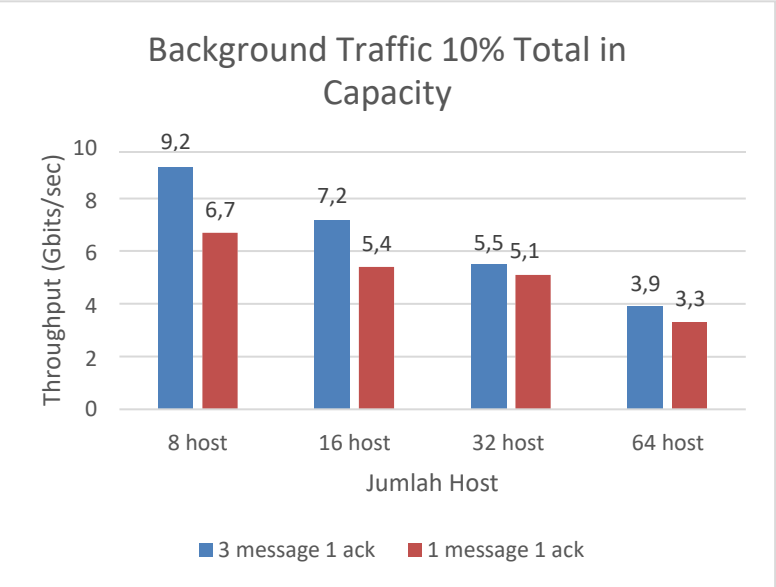
Pada grafik pada gambar 4.3., didapatkan hasil berupa nilai *throughput* metode *3 in 1* pada 16 *switch* berkisar 9-14,1 Gbits/sec dengan nilai rata-rata sebesar 13,19 Gbits/sec. Sedangkan nilai *throughput* satu pesan satu *acknowledgment* pada 16 *switch* berkisar pada 9,1-14,5 Gbits/sec dengan nilai rata-rata sebesar 12,1 Gbits/sec. Berdasarkan hasil tersebut, nilai *throughput* pada metode *3 in 1* dengan 16 *switch* memperoleh selisih 8,2% lebih tinggi dibandingkan dengan nilai *throughput* pada metode satu pesan satu *acknowledgment*.



Gambar 4. 4. Pengujian throughput tanpa background traffic sebanyak 32 *switch*

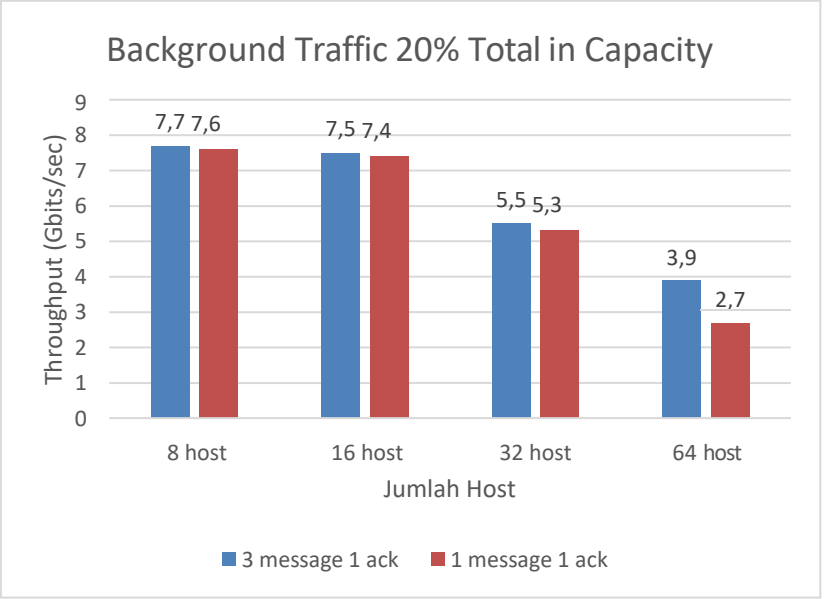
Pada grafik dalam gambar 4.4., didapatkan hasil berupa nilai *throughput* metode *3 in 1* pada 32 *switch* berkisar 4,2-9,3 Gbits/sec dengan nilai rata-rata sebesar 7,35 Gbits/sec. Sedangkan nilai *throughput* satu pesan satu *acknowledgment* pada 32 *switch* berkisar pada 5,5-8,4 Gbits/sec dengan nilai rata-rata sebesar 7,13 Gbits/sec. Berdasarkan hasil tersebut, nilai *throughput* pada metode *3 in 1* dengan 32 *switch* memperoleh selisih 2,9% lebih tinggi dibandingkan dengan nilai *throughput* pada metode satu pesan satu *acknowledgment*. Berdasarkan seluruh hasil selisih rata-rata pada masing-masing topologi dari 4 *switch* hingga 32 *switch*, metode *3 in 1* memperoleh nilai *throughput* yang lebih tinggi dibandingkan dengan metode satu pesan satu *acknowledgment*. Hal ini berpengaruh dikarenakan pengiriman pesan satu pesan satu *acknowledgment* lebih banyak proses pengiriman pesan dan menambah beban *controller* sehingga performansi nilai *throughput* menurun. Berbeda dengan proses *3 in 1* pengiriman pesan langsung dikirimkan tiga pesan dan dibalas dengan satu *acknowledgment* sehingga dapat mengurangi beban pada *controller*.

Pengujian kedua pada parameter *throughput* ini dilakukan dengan menggunakan *background traffic*. Pengujian dilakukan dengan cara yang sama seperti pengujian tanpa *background traffic*, namun dalam pengujian ini memiliki penambahan host lainnya yang menggunakan lalu lintas jaringan. Perbedaan yang dilakukan adalah pengiriman paket pada host utama dilakukan selama 10 detik sedangkan pengiriman paket pada host tambahan yang digunakan untuk *traffic* dilakukan selama 60 detik. Hal ini dimaksudkan agar ketika *host* utama mengirim paket, *host background traffic* akan tetap berjalan. Kemudian pada pengujian *background traffic* ini menggunakan beberapa jumlah *iperf streams* yang berbeda. Dalam hal ini bobot *background traffic* dibagi menjadi rentang 10-90%. Beban *traffic* 10-30% menggunakan jumlah *streams* sebanyak 1 *iperf streams*. Sedangkan mulai dari 40% hingga 90% jumlah *streams* ditambah 1, sehingga beban *traffic* pada 40% = 2; 50% = 3; 60% = 4; 70% = 5; 80% = 6, dan 90% = 7. Berikut adalah grafik pengujian nilai *throughput* menggunakan *background traffic.*



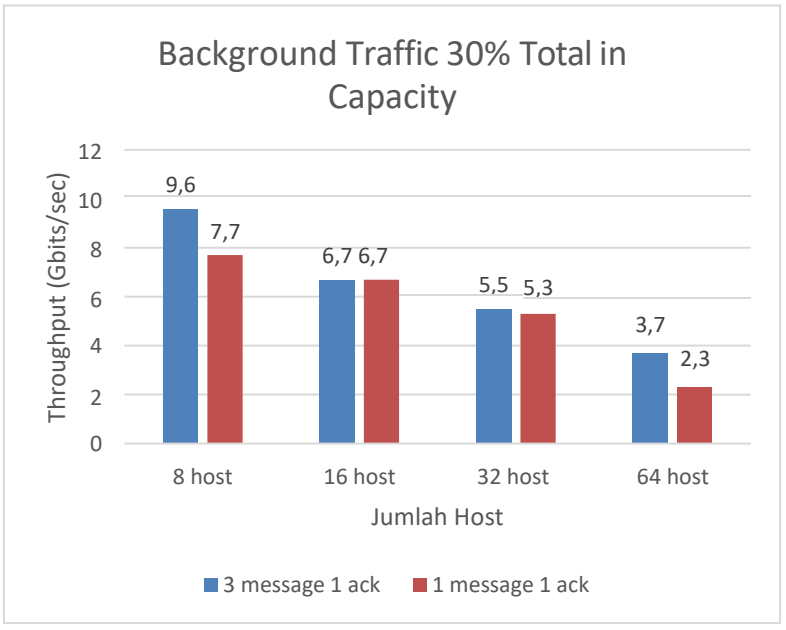
Gambar 4. 5. Pengujian *throughput* menggunakan *background traffic* 10%

Pada grafik dalam gambar 4.5., menunjukkan *throughput* pada *background traffic* 10% dengan nilai tertinggi pada 8 host yaitu pada topologi 4 *switch* dengan angka 9,2 Gbits/sec untuk metode *3 in 1* dan 6,7 Gbits/sec untuk metode satu pesan satu *acknowledgment.* Sedangkan nilai pada 64 host memperoleh angka terendah pada kedua metode. Jumlah *iperf streams* yang digunakan sebanyak 1 *stream.*



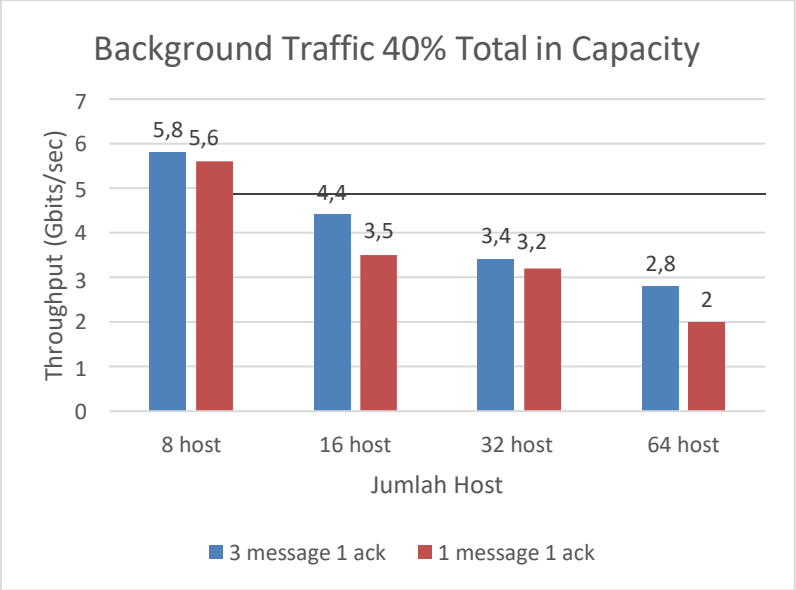
Gambar 4. 6. Pengujian *throughput* menggunakan *background traffic* 20%

Pada grafik dalam gambar 4.6., menunjukkan *throughput* pada *background traffic* 20% dengan nilai tertinggi pada 8 host yaitu pada topologi 4 *switch* dengan angka 7,7 Gbits/sec untuk metode *3 in 1*, kemudian 7,6 Gbits/sec untuk metode satu pesan satu *acknowledgment.* Sedangkan nilai pada 64 host memperoleh angka terendah pada kedua metode. Jumlah *iperf streams* yang digunakan sebanyak 1 *stream.*



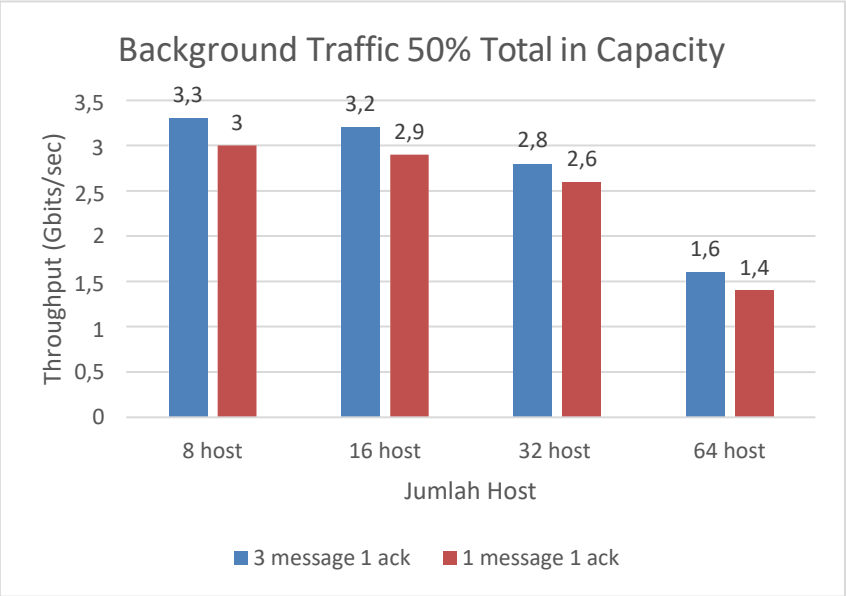
Gambar 4. 7. Pengujian *throughput* menggunakan *background traffic* 30%

Berdasarkan grafik pada gambar 4.7., menunjukkan *throughput* pada *background traffic* 30% dengan nilai tertinggi pada 8 host yaitu pada topologi 4 *switch* dengan angka 9,6 Gbits/sec untuk metode *3 in 1* dan 7,7 Gbits/sec untuk metode satu pesan satu *acknowledgment.* Sedangkan nilai pada 64 host memperoleh angka terendah pada kedua metode. Jumlah *iperf streams* yang digunakan sebanyak 1 *stream.*



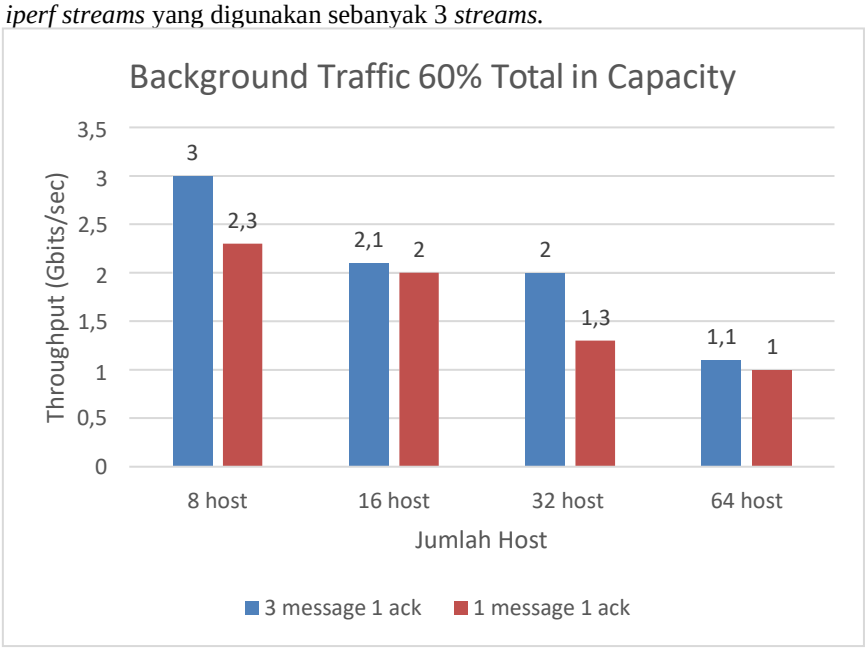
Gambar 4. 8. Pengujian *throughput* menggunakan *background traffic* 40%

Pada gambar 4.8., merupakan grafik yang menunjukkan nilai *throughput* pada *background traffic* 40% dengan nilai tertinggi pada 8 host yaitu pada topologi 4 *switch* dengan angka 5,8 Gbits/sec untuk metode *3 in1* dan 5,6 Gbits/sec untuk metode satu pesan satu *acknowledgment.* Sedangkan nilai pada 64 host memperoleh angka terendah pada kedua metode. Jumlah *iperf streams* yang digunakan sebanyak 2 *streams.*



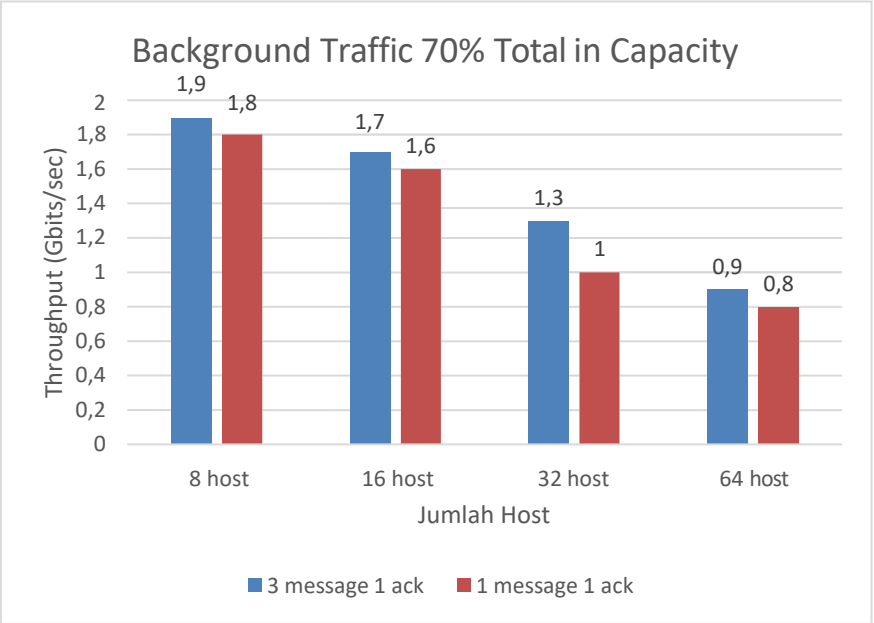
Gambar 4. 9. Pengujian *throughput* menggunakan *background traffic* 50%

Pada grafik dalam gambar 4.9., menunjukkan *throughput* pada *background traffic* 50% dengan nilai tertinggi pada 8 host yaitu pada topologi 4 *switch* dengan angka 3,3 Gbits/sec untuk metode *3 in 1* dan 3 Gbits/sec untuk metode satu pesan satu *acknowledgment.* Sedangkan nilai pada 64 host memperoleh angka terendah pada kedua metode. Jumlah *iperf streams* yang digunakan sebanyak 3 *streams.*



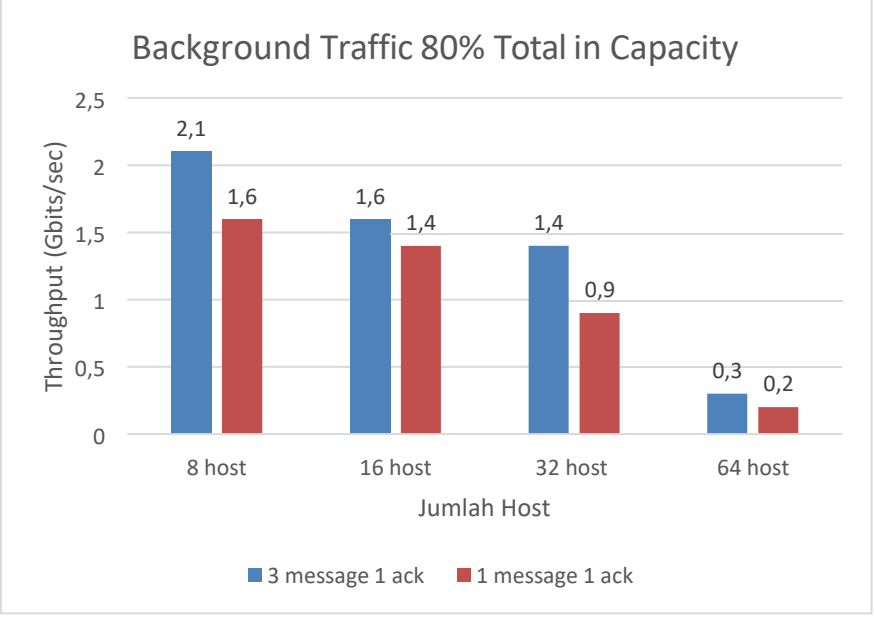
Gambar 4.10. Pengujian *throughput* menggunakan *background traffic* 60%

Berdasarkan grafik pada gambar 4.10., menunjukkan *throughput* pada *background traffic* 60% dengan nilai tertinggi pada 8 host yaitu pada topologi 4 *switch* dengan angka 3 Gbits/sec untuk metode *3 in 1*dan 2,3 Gbits/sec untuk metode satu pesan satu *acknowledgment.* Sedangkan nilai pada 64 host memperoleh angka terendah pada kedua metode. Jumlah *iperf streams* yang digunakan sebanyak 4 *streams.*



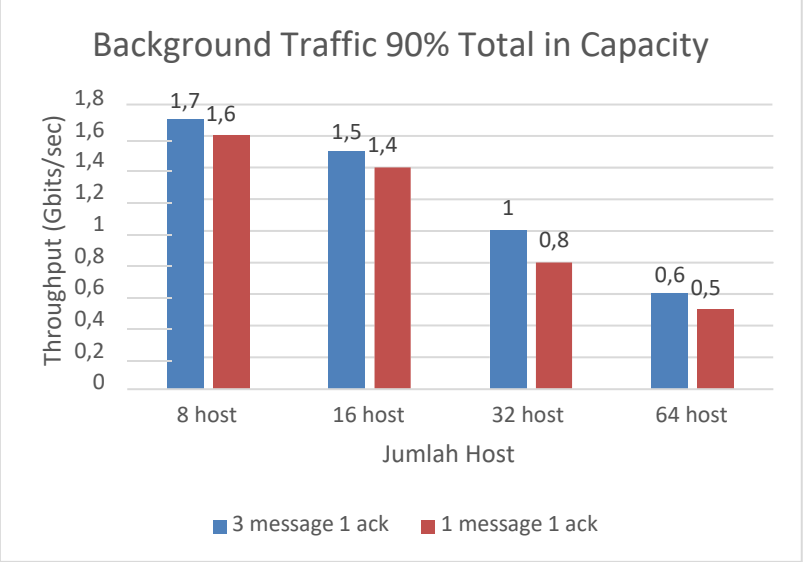
Gambar 4.11. Pengujian *throughput* menggunakan *background traffic* 70%

Berdasarkan grafik pada gambar 4.11., menunjukkan *throughput* pada *background traffic* 70% dengan nilai tertinggi pada 8 host yaitu pada topologi 4 *switch* dengan angka 1,9 Gbits/sec untuk metode *3 in 1* dan 1,8 Gbits/sec untuk metode satu pesan satu *acknowledgment.* Sedangkan nilai pada 64 host memperoleh angka terendah pada kedua metode. Jumlah *iperf streams* yang digunakan sebanyak 5 *streams.*



Gambar 4.12. Pengujian throughput menggunakan *background traffic* 80%

Pada grafik dalam gambar 4.12., menunjukkan *throughput* pada *background traffic* 80% dengan nilai tertinggi pada 8 host yaitu pada topologi 4 *switch* dengan angka 2,1 Gbits/sec untuk metode *3 in 1* dan 1,6 Gbits/sec untuk metode satu pesan satu *acknowledgment.* Sedangkan nilai pada 64 host memperoleh angka terendah pada kedua metode. Jumlah *iperf streams* yang digunakan sebanyak 6 *streams.*



Gambar 4.13. Pengujian throughput menggunakan *background traffic* 90%

Pada grafik dalam gambar 4.13., menunjukkan *throughput* pada *background traffic* 90% dengan nilai tertinggi pada 8 host yaitu pada topologi 4 *switch* dengan angka 1,7 Gbits/sec untuk metode *3 in 1* dan 1,6 Gbits/sec untuk metode satu pesan satu *acknowledgment.* Sedangkan nilai pada 64 host memperoleh angka terendah pada kedua metode. Jumlah *iperf streams* yang digunakan sebanyak 7 *streams.*

Berdasarkan hasil seluruh grafik *background traffic* dari 10% hingga 90%, kedua metode memperoleh hasil yang sama dengan nilai *throughput* tertinggi pada 8 *host* yaitu pada topologi 4 *switch.* Kemudian nilai *throughput* terendah terdapat pada 64 *host* yaitu pada topologi 32 *switch.* Grafik menunjukkan semakin banyak *switch* dan *host* yang dihubungkan, semakin menurunnya nilai *throughput.* Namun, pada metode *3 in 1* memperoleh nilai-nilai *throughput* yang cukup representatif lebih tinggi dibandingkan dengan satu pesan satu *acknowledgment.* Nilai-nilai tersebut juga berpengaruh dikarenakan penggunaan jumlah *iperf streams* yang berbeda-beda. Perbedaan yang diperoleh menunjukan bahwa semakin banyak jumlah *iperf streams* yang digunakan, nilai *throughput* semakin menurun. Namun, pada beban 10%-30% memperoleh nilai yang tidak jauh berbeda dikarenakan menggunakan jumlah *iperf streams* yang sama. Selain itu, penggunaan *iperf streams* terbanyak sebesar 7 *streams* dilakukan pada topologi 32 *switch* dengan 64 *host.* Semakin banyak jumlah *switch* yang terhubung maka semakin banyak beban yang dilakukan oleh *controller* untuk jalur *traffic* tersebut. Nilai *throughput* yang didapatkan menunjukkan penggunaan metode *3 in 1* memiliki kinerja yang lebih tinggi dibandingkan dengan metode satu pesan satu *acknowledgment.* Hal ini juga diperoleh karena berepengaruhnya dengan proses pengiriman pesan pada metode *3 in 1* yang lebih cepat dibandingkan dengan metode satu pesan satu *acknowledgment* dimana penggunaan *3 in 1* memiliki beban kerja yang lebih rendah pada *controller*. Dengan berkurangnya beban sumber daya pesan dalam melakukan pengiriman informasi pesan antara *controller* akan menghasilkan kinerja yang tinggi.

4.2. Uji Skenario Sebelum Terjadi *Down*

Pada pengujian skenario pertama yaitu sebelum *controller* terjadi *down* dilakukan pengujian fungsionalitas proses mekanisme *message exchange* pada *controller* 1 dan *controller* 2. Kemudian, pengujian selanjutnya meliputi nilai skalabilitas dengan parameter CPU *usage*. Pada pengujian ini dilakukan dengan membandingkan metode *3 in 1* dengan metode satu pesan satu *acknowledgment*. Kedua metode tersebut menggunakan *distributed controller* (*active-active*).

```
[log] routing_key : 'swIdController1', message_topic : 'msgController1', body message : '1'
[log] routing_key : 'messageAckB', message_topic : 'msgAckController2', body message : 'ACK From Controller B'
[log] routing_key : 'swIdController1', message_topic : 'msgController1', body message : '2'
[log] routing_key : 'messageAckB', message_topic : 'msgAckController2', body message : 'ACK From Controller B'
[log] routing_key : 'swIdController1', message_topic : 'msgController1', body message : '3'
[log] routing_key : 'messageAckB', message_topic : 'msgAckController2', body message : 'ACK From Controller B'
[log] routing_key : 'swIdController1', message_topic : 'msgController1', body message : '4'
[log] routing_key : 'messageAckB', message_topic : 'msgAckController2', body message : 'ACK From Controller B'
```

Gambar 4.14. Proses *message exchange* satu pesan satu *acknowledgment*

Gambar 4.14. merupakan proses *message exchange* yang terjadi pada metode satu pesan satu *acknowledgment*. Proses dilakukan ketika mengirim pesan akan langsung dibalas oleh *controller* 2 kemudian dapat melakukan proses pengiriman pesan berikutnya.

```
[log] routing_key : 'swIdController1', message_topic : 'msgController1', body message : '4'
[log] routing_key : 'swIdController1', message_topic : 'msgController1', body message : '1'
[log] routing_key : 'swIdController1', message_topic : 'msgController1', body message : '2'
[log] routing_key : 'messageAckB', message_topic : 'msgAckController2', body message : 'ACK From Controller B'
[log] routing_key : 'swIdController1', message_topic : 'msgController1', body message : '3'
[log] routing_key : 'swIdController1', message_topic : 'msgController1', body message : '4'
[log] routing_key : 'swIdController1', message_topic : 'msgController1', body message : '1'
[log] routing_key : 'messageAckB', message_topic : 'msgAckController2', body message : 'ACK From Controller B'
[log] routing_key : 'swIdController1', message_topic : 'msgController1', body message : '2'
[log] routing_key : 'swIdController1', message_topic : 'msgController1', body message : '3'
[log] routing_key : 'swIdController1', message_topic : 'msgController1', body message : '4'
[log] routing_key : 'messageAckB', message_topic : 'msgAckController2', body message : 'ACK From Controller B'
[log] routing_key : 'swIdController1', message_topic : 'msgController1', body message : '1'
[log] routing_key : 'swIdController1', message_topic : 'msgController1', body message : '2'
[log] routing_key : 'swIdController1', message_topic : 'msgController1', body message : '3'
[log] routing_key : 'messageAckB', message_topic : 'msgAckController2', body message : 'ACK From Controller B'
```

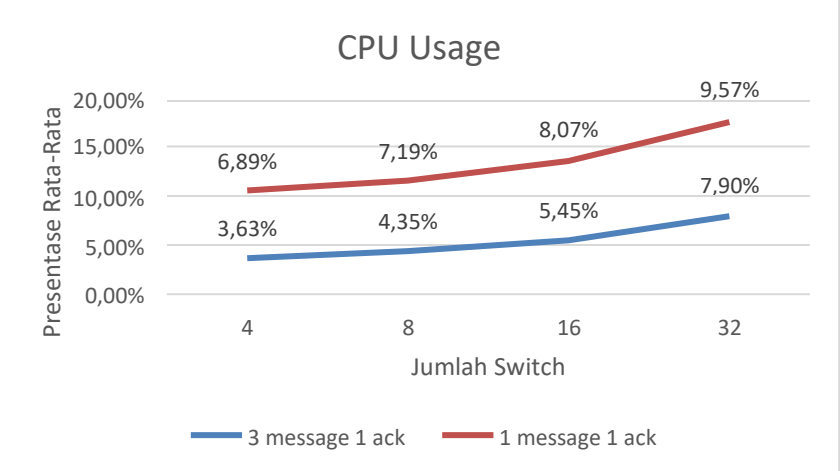
Gambar 4.15. Proses *message exchange 3 in 1*

Gambar 4.15. merupakan proses *message exchange* dengan metode *3 in 1* sebelum terjadi *down.* Pengiriman pesan dilakukan dari *controller* 1 sebanyak tiga pesan kemudian dibalas oleh *controller* 2 dengan satu *acknowledgment.* Proses pengiriman pesan dilakukan 1 detik jika melebihi waktu tersebut *controller* akan terjadi *down* dan akan melakukan *switch migration* terhadap *controller* 2.

Berdasarkan gambar tersebut menunjukkan metode *3 in 1* memiliki proses pengiriman pesan yang lebih cepat dibandingkan dengan satu pesan satu *acknowledgment*. Sehingga mengurangi beban sumber daya pesan dalam melakukan pengiriman informasi pesan antar *controller*.

4.2.1. Pengujian CPU *Usage*

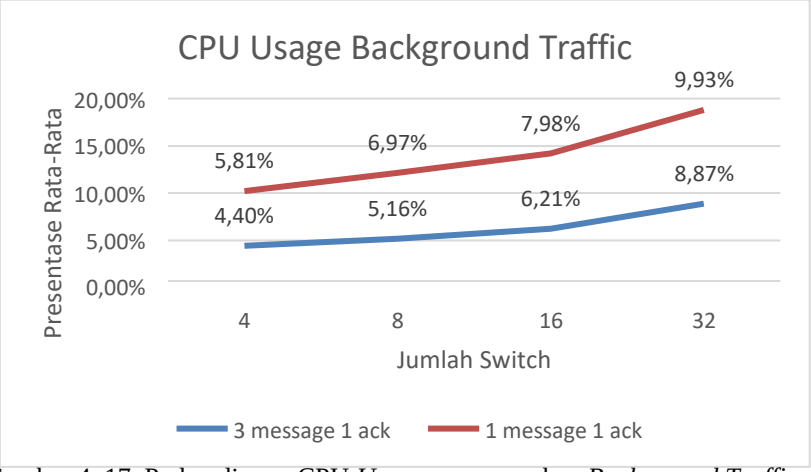
Pengujian CPU *Usage* dilakukan berdasarkan dua cara yang pertama adalah tanpa *background traffic* dan pengujian kedua menggunakan *background traffic.* Pengujian nilai CPU *usage* bertujuan untuk merepresentasikan skalabilitas jaringan yang berpengaruh terhadap beban kerja pada *controller.* Berdasarkan hal tersebut dilakukan perbandingan antara metode *3 in 1* dan satu pesan satu *acknowledgment.*



Gambar 4. 16. Perbandingan CPU *Usage* tanpa *Background Traffic*

Pada gambar 4.16., merupakan hasil rata-rata perbandingan dari nilai CPU *Usage* pada kedua metode yaitu metode *3 in 1* dengan metode satu pesan satu *acknowledgment*. Grafik tersebut cukup representatif dari 4 *switch*

hingga 32 *switch* yang semakin meningkat. Pengujian dilakukan sebanyak 10 kali untuk mendapatkan hasil tersebut dengan waktu pengujian selama 10 detik untuk masing-masing *switch*. Nilai presentase tertinggi yang didapatkan pada metode *3 in 1* sebesar 7,90% sedangkan nilai tertinggi pada metode satu pesan satu *acknowledgement* sebesar 9,57%. Berdasarkan hasil tersebut metode *3 in 1* memperoleh nilai selisih 1,67% lebih rendah dibandingkan dengan metode satu pesan satu *acknowledgment*. Sehingga beban kerja pada metode *3 in 1* pun memiliki beban kerja yang lebih rendah dibandingkan dengan metode satu pesan satu *acknowledgment*. Hal tersebut dikarenakan pengiriman pesan langsung dikirimkan tiga pesan dan dibalas dengan satu *acknowledgment* yang berisi tiga pesan tersebut sehingga beban kerja pada *controller* berkurang. Kemudian, semakin banyak jumlah *switch* yang digunakan maka CPU *Usage* akan semakin tinggi dikarenakan banyaknya *switch* yang terhubung pada *controller*.

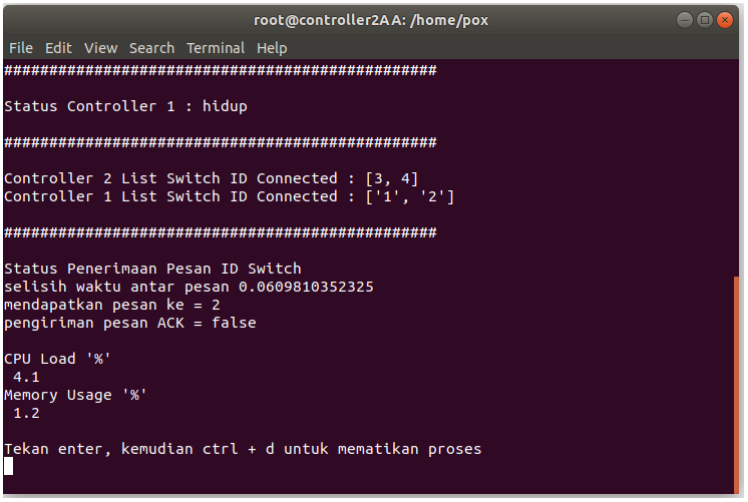


Gambar 4. 17. Perbandingan CPU *Usage* menggunakan *Background Traffic*

Berdasarkan grafik pada gambar 4.17., CPU *Usage* menggunakan *background traffic* penggunaan kedua metode memperoleh hasil yang sama seperti sebelumnya. Metode *3 in 1* memiliki hasil yang lebih rendah dibandingkan dengan metode satu pesan satu *acknowledgment*. Hal tersebut dikarenakan pengiriman pesan yang memiliki proses lebih cepat sangat berpengaruh pada beban kerja pada *controller*.

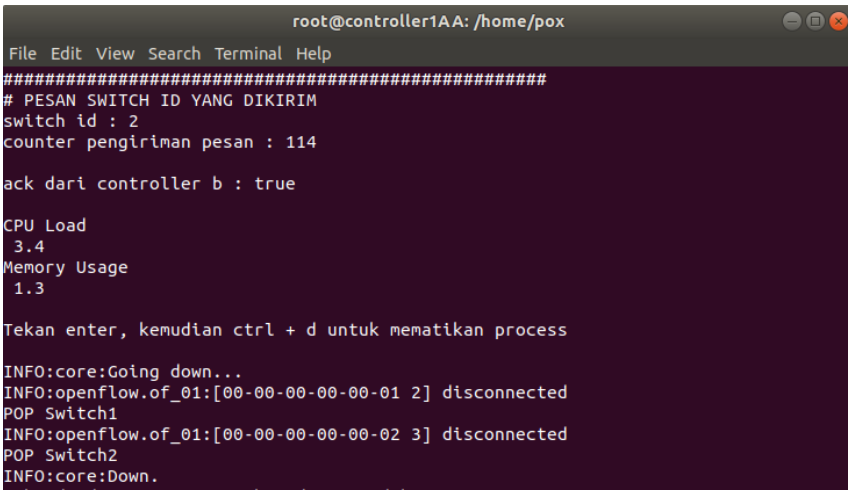
4.3. Uji Skenario Terjadi *Down*

Pengujian skenario kedua yang dilakukan yaitu ketika terjadinya *down* pada salah satu *controller*. Penelitian ini dilakukan menggunakan *distributed controller* (*active-active*) sehingga kedua *controller* saling aktif bekerja sama. Pengujian dilakukan dengan cara mematikan *controller* 1. Ketika *controller* 1 sudah *down* maka *RabbitMQ* sebagai *message broker* akan menerima pesan bahwa *controller* 1 *down* dan melakukan *forward* kepada *controller* 2, kemudian akan melakukan *switch migration* dari *controller* 1 ke controller 2. *Switch* yang pada awalnya terhubung pada controller 1 akan berpindah ke controller 2.



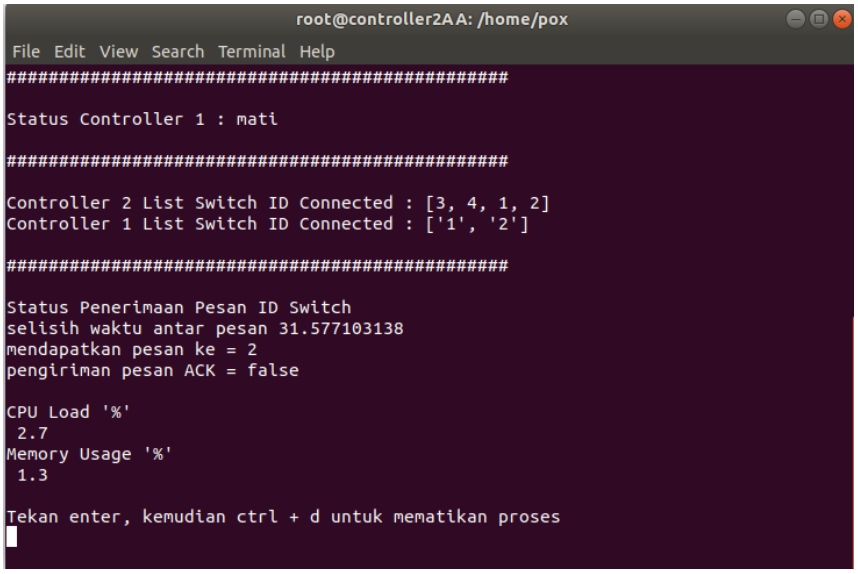
Gambar 4. 18. Kondisi *Controller* 2 sebelum *Controller* 1 terjadi *down*

Pada gambar 4.18., merupakan kondisi awal sebelum *controller* 1 terjadi *down*. *Switch* 1 dan *switch* 2 masih terhubung dengan *controller* 1, dan *controller* 2 mendeteksi status *controller* 1 hidup.



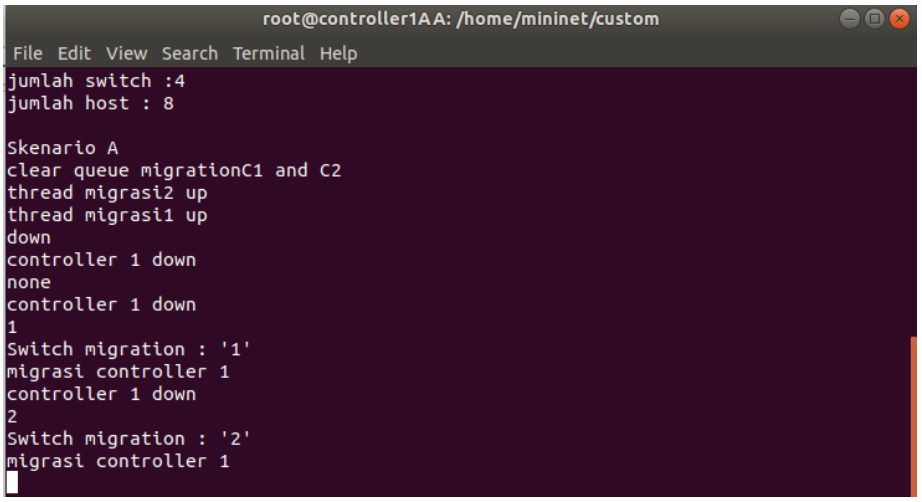
Gambar 4. 19. Kondisi *Controller* 1 *down*

Pada gambar 4.19., merupakan kondisi *controller* 1 ketika terjadi *down*. *Switch* 1 dan *switch* 2 sudah tidak terhubung (*disconnected*) dengan *controller* 1.



Gambar 4. 20. Kondisi *controller* 2 ketika *controller* 1 *down*

Pada gambar 4.20. merupakan kondisi *controller* 2 yang telah mendeteksi bahwa *controller* 1 terjadi *down*. Kemudian *switch* yang sebelumnya terhubung dengan *controller* 1 telah berpindah (*switch migration*) ke *controller* 2.

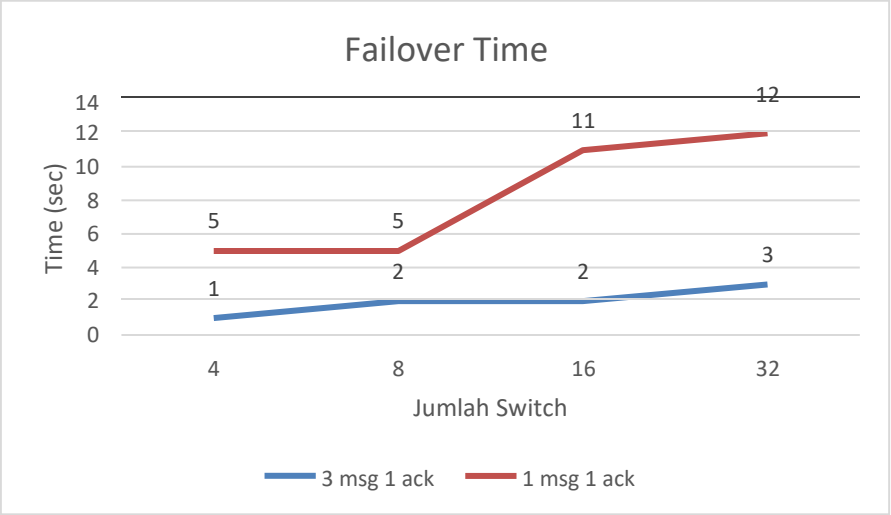


Gambar 4.21., kondisi mininet ketika *controller* 1 terjadi *down*

Gambar 4.21., merupakan kondisi mininet ketika terjadi *controller* 1 terjadi *down* dan melakukan *switch migration* dari *controller* 1 terhadap *controller* 2.

4.3.1. Pengujian Waktu Failover

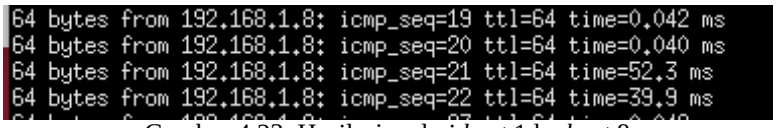
Pada pengujian waktu terjadinya mekanisme *failover* dilakukan dengan cara mematikan proses pada *controller* 1 sehingga terjadi *down*. Waktu *failover* akan dibandingkan dengan kedua metode yang diuji yaitu metode *3 in 1* dan satu pesan satu *acknowledgment*. Untuk melihat selisih waktu ketika *switch-switch* yang sudah tidak terhubung dengan *controller* 1 hingga adanya informasi *switch* berpindah ke *controller* 2 adalah dengan menjalankan perintah *tail -f /var/log/openvswitch/ovs-vswitchd.log*.



Gambar 4.22. Selisih waktu *failover*

Pada gambar 4.22., merupakan grafik yang menunjukkan selisih waktu *failover* untuk melakukan *switch migration* ketika *controller* 1 terjadi *down*. Selisih tersebut didapatkan dari waktu kondisi *switch* yang sudah tidak terhubung (*disconnected*) pada *controller* 1 hingga waktu terhubungnya semua *switch* tersebut pada *controller* 2 dengan stabil. Waktu *failover* berbanding lurus dengan jumlah *switch* yang dikendalikan oleh *controller*. Grafik tersebut menjelaskan bahwa semakin banyak jumlah *switch*, maka waktu *failover* pun akan semakin tinggi.

Hasil pengujian menunjukkan waktu *failover* pada metode *3 in 1* lebih efisien karena lebih cepat ditanggap oleh *controller* 2. Hal tersebut berpengaruh dengan beban kerja *controller* yang berkurang dikarenakan proses pengiriman pesan *3 in 1* lebih cepat dengan langsung mengirimkan tiga pesan dan dibalas satu *acknowledgment*.



Gambar 4.23. Hasil *ping* dari *host* 1 ke *host* 8

Pada gambar 4.23., merupakan hasil yang diperoleh ketika melakukan *ping* dari *host* 1 ke *host* 8 selama proses *failover* berlangsung. Gambar tersebut menjelaskan bahwa setelah "icmp_seq=20", *switch* tidakterhubung dengan *controller* 1, sehingga proses *ping* mengalami *delay* dan pada waktu tersebut *switch* mencoba untuk menghubungkan ke *controller* 2. Hal ini terjadi karena *RabbitMQ* sebagai *message broker* membutuhkan waktu untuk melakukan *switch migration* dari *controller* 1 ke *controller* 2.

5. Kesimpulan

Berdasarkan hasil pengujian yang telah dilakukan dari perbandingan metode *3 in 1* yaitu tiga pesan satu *acknowledgment* dan metode satu pesan satu *acknowledgment* memperoleh nilai *throughput* metode *3 in 1* lebih tinggi, baik tanpa *background traffic* maupun menggunakan *background traffic*. Hal tersebut berpengaruh karena semakin banyak jumlah *switch* yang terhubung maka semakin banyak beban yang dilakukan oleh *controller* untuk jalur *traffic* tersebut. Hal ini juga didapatkan karena berepengaruhnya pada proses pengiriman pesan pada metode *3 in 1* yang lebih cepat dibandingkan dengan metode satu pesan satu *acknowledgment* dimana penggunaan *3 in 1* yang memiliki beban kerja yang lebih rendah pada *controller*. Oleh karena itu, dengan berkurangnya beban sumber daya pesan dalam melakukan pengiriman informasi pesan antara *controller* akan menghasilkan kinerja yang lebih tinggi.

Pada pengujian CPU *Usage* dengan penggunaan *background traffic* memperoleh hasil yang sama seperti tanpa menggunakan *background traffic*, nilai yang didapatkan metode *3 in 1* lebih rendah dibandingkan satu pesan satu *acknowledgment.* Oleh karena itu, beban kerja tiap *controller* pada metode *3 in 1* berkurang. Pada pengujian waktu *failover,* penggunaan metode *3 in 1* memperoleh waktu yang lebih cepat untuk menerima migrasi *switch* dari *controller* 1. Hal tersebut berpengaruh karena CPU *Usage* yang lebih rendah dengan berkurangnya sumber daya pesan yang dilakukan ketika melakukan pengiriman pesan sehingga proses *failover* berjalan lebih cepat. Hasil pengujian menyimpulkan bahwa metode *3 in 1* mengungguli dari metode satu pesan satu *acknowledgment.* Hasil membuktikan metode *3 in 1* memiliki peningkatan kinerja serta beban kerja pada *controller* yang lebih rendah. Selain itu, proses *failover* menghasilkan waktu yang lebih cepat. Namun, pada penelitian ini hanya dilakukan dengan menggunakan dua *controller* dimana *controller* 1 hanya mengirimkan pesan dan *controller* 2 bertugas untuk membalas pesan. Oleh karena itu, saran untuk penelitian selanjutnya dapat menjadikan *controller* 2 sebagai *controller* utama, sehingga yang semulanya *controller* 1 terjadi *down* kemudian *controller* tersebut hidup kembali, *controller* 2 akan menjadi *controller* utama yang bertugas untuk mengirimkan tiga pesan dan *controller* 1 yang akan membalas pesan. Adapun saran lainnya yaitu melakukan penambahan *controller* menggunakan metode *3 in 1* sehingga pengiriman pesan tidak hanya dilakukan oleh satu *controller.*

Daftar Pustaka

[1] H. G. Ahmed and R. Ramalakshmi, "Performance Analysis of Centralized and Distributed SDN Controllers for Load Balancing Application," 2018 2nd Int. Conf. Trends Electron. Informatics, no. Icoei, pp. 758–764, 2018.
[2] O. Blial, M. Ben Mamoun, and R. Benaini, "An Overview on SDN Architectures with Multiple Controllers," J. Comput. Networks Commun., vol. 2016, 2016.
[3] M.N.A. Syaehoni, "Desain Distributed Controller dengan Metode Active-Active pada Jaringan Software Define Network," Program Studi Sarjana Informatika Fakultas Informatika Universitas Telkom Bandung, 2019.
[4] E. S. Spalla et al., "AR2C2: Actively replicated controllers for SDN resilient control plane," Proc. NOMS 2016 - 2016 IEEE/IFIP Netw. Oper. Manag. Symp., no. Noms, pp. 189–196, 2016.
[5] K. S. Sahoo, S. Mohanty, M. Tiwary, B. K. Mishra, and B. Sahoo, "A comprehensive tutorial on software defined network: The driving force for the future internet technology," ACM Int. Conf. Proceeding Ser., vol. 12-13-August-2016, no. August, 2016.
[6] M. H. Hidayat and N. R. Rosyid, "Analisis Kinerja dan Karakteristik Arsitektur Software-Defined Network Berbasis OpenDaylight Controller," Citee, no. 2085–6350, pp. 194–200, 2017.
[7] T. F. Oliveira and L. F. Q. Silveria, "Distributed SDN controllers optimization for energy saving," 2019 Fourth Int. Conf. Fog Mob. Edge Comput., pp. 86–89, 2019.
[8] I. Z. Bholebawa and U. D. Dalal, "Design and Performance Analysis of OpenFlow-Enabled Network Topologies Using Mininet," Int. J. Comput. Commun. Eng., vol. 5, no. 6, pp. 419–429, 2016.
[9] T. Kim, T. Koo, and E. Paik, "SDN and NFV benchmarking for performance and reliability," 17th Asia-Pacific Netw. Oper. Manag. Symp. Manag. a Very Connect. World, APNOMS 2015, pp. 600–603, 2015.

[10] F. Fatturrahman, “SDN Controller Robustness and Distribution Framework,” 2017.
[11] A. Kondel, “Evaluating System Performance for handling scalability challenge in SDN,” pp. 594–597, 2015.
[12] A. Abdelhafez, E. Alba, and G. Luque, “Performance analysis of synchronous and asynchronous distributed genetic algorithms on multiprocessors,” Swarm Evol. Comput., vol. 49, no. June, pp. 147–157, 2019.
[13] F. Benamrane, M. Ben Mamoun, and R. Benaini, “New method for controller-to-controller communication in distributed SDN architecture,” Int. J. Commun. Networks Distrib. Syst., vol. 19, no. 3, pp. 357–367, 2017.
[14] M. Nathalia, “Analisis Unjuk Kerja TCP Reno di Jaringan Single HOP Wireless Link,” Program Studi Teknik Informatika Fakultas Sains dan Teknologi Universitas Sanata Dharma Yogyakarta, 2016.