Analisis *Robustness Controller* Terhadap Skalabilitas Jaringan Pada *Software-Defined Network* dengan Model *Single Controller* dan *Flat Controller*

Deryandaru Bhaskara[1], Vera Suryani[2], Muhammad Arief Nugroho[3]

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

Abstrak

Bertambahnya *switch* dan *host* pada suatu jaringan dapat meningkatan *load* pada *controller* yang mengakibatkan penurunan kualitas kinerja jaringan. Pada model *single controller* terdapat permasalahan *robustness controller* ketika menerapkan skalabilitas jaringan. Hal tersebut menyebabkan penurunan kinerja jaringan karena *load* pada *controller* yang semakin besar. Pada penelitian ini diusulkan model *flat controller(active-active)* yang memungkinkan kedua *controller* untuk bekerja secara bersamaan. Selain itu, pada penelitian ini juga digunakan metode *clustering* untuk menghubungkan kedua *controller*, sehingga dapat menurunkan *load* pada kedua *controller* dan menghasilkan kualitas kinerja jaringan yang lebih baik. Pada pelaksanaan simulasi digunakan emulator mininet untuk membangun topologi jaringan SDN dan ONOS sebagai *controller* yang terhubung dengan jaringan tersebut. Berdasarkan uji perbandingan permformansi pada model *single controller* dan *flat controller(active-active)*, diketahui model *flat controller(active-active)* mempengaruhi kinerja jaringan dengan selisih *throughput* 23.9% lebih besar dan selisih *latency* 11.45% lebih rendah. Pada pengujian penggunaan CPU dengan penambahan jumlah *background traffic*, model *flat controller(active-active)* menghasilkan penggunaan CPU 1.07% lebih rendah dibandingkan dengan *single controller*. Berdasarkan hasil pengujian, dapat disimpulkan bahwa model *single controller* memiliki *robustness controller* yang lebih baik terhadap skalabilitas jaringan dibandingkan dengan model *flat controller(active-active)*.

Kata kunci: *single controller, flat controller, robustness, clustering*

Abstract

The addition of switches and hosts on a network can increase the load on the controller which results in a decrease in the quality of network performance. In the single controller model, there is a robustness controller problem when implementing network scalability. This causes a decrease in network performance because the load on the controller is getting bigger. This research proposes a flat controller (active-active) model that allows both controllers to work simultaneously. In addition, in this study, the clustering method is also used to connect the two controllers, so that it can reduce the load on both controllers and produce better quality network performance. In the implementation of the simulation, the mininet emulator is used to build the SDN and ONOS network topology as a controller connected to the network. Based on the performance comparison test on the single controller and flat controller (active-active) models, it is known that the flat controller (active-active) model affects network performance with a 23.9% greater difference in throughput and 11.45% lower latency difference. In testing CPU usage with an increase in the amount of background traffic, the flat controller (active-active) model resulted in 1.07% lower CPU usage than the single controller. Based on the test results, it can be concluded that the single controller model has a better robustness controller for network scalability than the flat controller (active-active) model.

Keywords: single controller, flat controller, robustness, clustering

1. Pendahuluan

Latar Belakang

Berdasarkan model *controller*, arsitektur SDN dibagi menjadi dua, yaitu *single controller* dan *distributed controller*. *Single controller* merupakan model *controller* yang terdiri satu *controller* utama yang menghubungkan seluruh *switch* yang ada pada suatu arsitektur jaringan SDN [1]. Model tersebut memiliki permasalahan utama, yaitu *robustness* pada *controller*. Permasalahan *robustness* dapat disebabkan oleh meningkatnya *traffic* jaringan dan *load controller* yang tinggi. Tingginya *load controller* yang disebabkan oleh meningkatnya jumlah perangkat jaringan dapat

mengakibatkan menurunnya kualitas kinerja jaringan [2]. Permasalahan tersebut dapat diatasi dengan metode *distributed controller* yang membagi *load controller* menjadi sama rata ke sejumlah controller.

Model yang digunakan pada penelitian ini yaitu, model *flat controller*. Dalam model ini, beberapa *controller* memungkinkan untuk dapat memiliki tanggung jawab yang sama terhadap seluruh jaringan dengan membagi *load* sama rata di setiap *controller* [3]. Pemilihan *flat controller* sebagai bentuk arsitektur model *distributed controller* pada penelitian ini adalah kemampuannya yang dapat memperluas kapabilitas dari *control plane*. Jika dibandingkan dengan *hierarchical controller*, *flat controller* memiliki tingkat komunikasi antar *controller* yang lebih tinggi untuk menjamin pengelolaan jaringan yang lebih konsisten [4]. Peran pada *controller* dapat dibagi menjadi dua, yaitu *active-active* dan *active-backup.* Pada penelitian ini diterapkan peran *controller active-active* yang memungkinkan seluruh *controller* aktif secara bersamaan untuk mengelola seluruh jaringan. Berdasarkan penelitian sebelumnya, pengujian skalabilitas jaringan pada model *flat controller(active-active)* menghasilkan nilai *throughput* yang lebih tinggi jika dibandingkan dengan *flat controller(active-backup)* [5]. Selain itu, pada model *flat controller* juga diterapkan metode *clustering* yang bertujuan untuk menghubungkan kedua *controller* dan dapat mengelola seluruh jaringan secara bersamaan. Metode ini berfungsi untuk meringankan *load* pada kedua *controller*, sehingga dapat menghasilkan kinerja jaringan yang lebih baik [6].

Topik dan Batasannya

Skalabilitas jaringan menjadi permasalahan pada model *single controller* karena dapat menyebabkan penurunan kinerja jaringan yang disebabkan oleh tingginya *load* pada *controller* [2]. Sebuah jaringan dapat dikatakan *robust* adalah ketika komunikasi antara dua buah node tidak terpengaruh oleh kegagalan jaringan. Kemungkinan kegagalan jaringan atau menurunnya kinerja jaringan ini dapat diatasi dengan mengurangi jumlah node yang terhubung pada sebuah *controller* [7]. Oleh karena itu, solusi yang digunakan untuk mengatasi permasalahan tersebut adalah dengan menggunakan *distributed controller* dengan model *flat controller* dan menerapkan metode *clustering* pada *controller* untuk menggabungkan dua buah *cluster* menjadi satu.

Batasan masalah pada penelitian ini, yaitu melakukan pengujian kinerja jaringan dengan parameter *throughput* dan *latency* untuk menganalisis dampak skalabilitas jaringan terhadap kualitias kinerja jaringan, pengujian penggunaan CPU dan *memory* untuk menganalisis *load* pada *controller*, serta pengujian reliabilitas untuk mengetahui dampak penempatan *controller* yang sejajar pada model *flat controller* yang dilakukan dengan pengujian *latency*. Selain itu, pada penelitian ini digunakan mininet sebagai emulator untuk simulasi jaringan, ONOS dengan versi 1.13.2 sebagai jenis *controller* yang digunakan, dan penggunaan tiga buah *virtualbox* untuk mengoperasikan mininet dan dua *controller*.

Tujuan

Berdasarkan pada hal-hal yang telah dijelaskan di atas, penelitian ini bertujuan untuk melakukan simulasi dan menganalisis penggunaan model *flat controller (active-active)* sebagai metode untuk permasalahan *robustness* pada model *single controller* yang disebabkan oleh skalabilitas jaringan.

Organisasi Tulisan

Bagian selanjutnya terdiri atas studi terkait, sistem yang dibangun, evaluasi, dan kesimpulan. Pada bagian studi terkait akan dijelaskan studi literatur atau teori pendukung yang menjadi dasar pelaksanaan penelitian ini. Pada bagian sistem yang dibangun akan dijelaskan perancangan dan implementasi sistem. Dari implementasi sistem tersebut akan dilaksanakan pengujian terhadap sistem berdasarkan beberapa skenario pengujian. Setelah itu, hasil dan analisis pengujian tersebut akan ditulis pada bagian evaluasi. Lalu, pada bagian kesimpulan akan dituliskan konklusi dari hasil dan analisis pengujian pada bagian evaluasi.

2. Studi Terkait

2.1. Related Works

Terdapat beberapa jurnal atau paper yang telah dibahas terkait penelitian yang dilakukan diantaranya seperti pada tabel berikut.

Tabel 2.1. *Related Works*

Judul	Metode	Hasil	Keterangan
An Overview on SDN Architectures with Multiple	Perbandingan model *Flat Controller* dengan *Hierarchical*	Model *hierarchical controller* memiliki risiko kegagalan *controller* karena bentuk *controller* yang tersentralisasi pada	Penggunaan model *flat controller* pada penelitian ini disebabkan oleh kelebihan yang terdapat pada model ini yang dapat mengurangi

Controllers (2016)	*Controller*.	root controller, sedangkan model *flat controller* lebih tahan terhadap kegagalan *controller* karena semua *controller* berada pada *layer* yang sama [3].	tingkat risiko kegagalan *controller* karena diletakkan pada posisi yang sama untuk setiap *controller*, sehingga beban kerja *controller* dapat dibagi rata.
Distributed Controller Clustering in Software Defined Networks (2017)	Pengujian kinerja jaringan pada *distributed controller* yang menggunakan *clustering* dan tanpa *clustering*.	Selain menghasilkan penggunaan CPU yang lebih baik, kinerja jaringan *distributed controller* dengan menggunakan *clustering* juga menghasilkan penurunan tingkat *delay* dari 8.1% menjadi 1.6% dan penurunan tingkat *packet loss* dari 5.22% menjadi 4.15% [6].	Pada paper tersebut telah disebutkan bahwa terdapat penurunan kualitas kinerja jaringan saat terjadi penambahan *host* dan *switch* pada jumlah tertentu. Oleh karena itu, pada penelitian ini diterapkan metode *clustering* untuk mengurangi beban kerja pada *controller*, sehingga dapat meningkatkan kualitas kinerja jaringan.
A Comparative Evaluation of The Performance of Popular SDN Controllers (2018)	Perbandingan hasil *throughput* dari *controller* Ryu, ONOS, Floodlight, dan OpenDayLight.	Setelah melakukan pengujian dengan jumlah *switch* yang berbeda, ONOS menghasilkan nilai *throughput* tertinggi, jika dibandingkan dengan *controller* lainnya. Selain itu, ONOS juga menghasil nilai *throughput* yang relatif stabil saat jumlah *switch* lebih dari 10 [8].	Berdasarkan hasil pada paper tersebut, pada penelitian ini digunakan *controller* ONOS untuk menghasilkan nilai *throughput* yang relatif stabil saat diterapkan skalabilitas jaringan.
Desain *Distributed Controller* dengan Metode *Active-Active* pada Jaringan *Software Define Network* (2019)	Perbandingan *distributed controller* dengan *active-active* dan *active-backup* yang diuji dengan skalabilitas jaringan.	Pengujian dengan menggunakan metode *active-active* menghasilkan nilai *throughput* yang lebih tinggi dibandingkan dengan metode *active-backup* [5].	Pada penelitian ini digunakan metode *active-active* pada *flat controller* karena dengan aktifnya kedua *controller* dapat mengurangi beban kerja dari sebuah *controller*. Hal ini memungkinkan kualitas jaringan yang lebih baik jika dibandingkan dengan metode *active-backup* yang menjadikan salah satu *controller* saja yang bekerja dalam satu waktu.
Simulasi Kinerja Berbagai Topologi Jaringan Berbasis *Software-Defined Network* (SDN) (2017)	Perbandingan kualitas kinerja jaringan SDN dengan parameter *Quality of Service (QoS)* pada topologi *linier*, *tree*, *star*, *ring*, dan *full mesh*.	Pengujian pada kualitas kinerja jaringan dengan topologi *linier* menghasilkan nilai *delay*, *jitter*, dan *throughput* yang lebih baik dibandingkan topologi lainnya [9].	Berdasarkan hasil pada penelitian tersebut, pada penelitian ini digunakan topologi *linier* untuk menghasilkan nilai kualitas kinerja jaringan yang lebih baik.

2.2. Software-Defined Network

Software-Defined Network adalah paradigma jaringan yang baru berkembang dengan memberikan harapan untuk mengubah keterbatasan yang ada pada infrastruktur jaringan saat ini. Pada konsep SDN, pemisahan *control plane* yang mendasari fungsi router dan *data plane* sebagai fungsi *switch* yang melakukan *forwarding traffic* dilakukan. Setelah pemisahan *control plane* dan *data plane* dilakukan, perangkat *switch* menjadi *forwarding device* yang lebih sederhana dan *control plane* diimplementasikan pada *controller* yang tersentralisasi dengan konfigurasi sistem operasi jaringan [10]. *Controller* memrogram *forwarding device* untuk menambahkan aksi kepada paket yang akan diteruskan, seperti *drop* paket, teruskan paket ke port tertentu, teruskan paket ke *controller*, dan *rewrite header* pada paket. Semua instruksi tersebut diaplikasikan oleh *switch* yang dikelola oleh *controller* melalui protokol OpenFlow [10]. Berdasarkan

model *controller*, *Software-Defined Network* dapat dibagi menjadi dua, yaitu *single controller* dan *distributed controller*.

2.2.1. *Single Controller*

Single controller merupakan sebuah model arsitektur SDN yang terdiri dari satu *controller*. Semua pemrosesan yang ada pada *control plane* akan diproses pada satu *controller* tersebut [11]. *Controller* dengan mudah mengontrol semua *switch* secara terpusat karena terhubung ke satu titik, yaitu satu *controller* tersebut [12]. Namun, pada jenis *controller* ini terdapat kekurangan pada kemampuan *controller* untuk mengatasi skalabilitas jaringan. Hal tersebut dapat menyebabkan terhambatnya kinerja *controller* karena jumlah *load* yang semakin membesar dan tugas untuk mengelola jaringan semakin luas [2].

2.2.2. *Distributed Controller*

Distributed Controller merupakan sebuah arsitektur *multicontroller* yang memiliki beberapa *controller* dan saling bekerja sama untuk mencapai level kinerja yang lebih baik dan memiliki skalabilitas. Software-defined network memiliki sebuah arsitektur yang disebut sebagai *physically distributed*. Arsitektur ini merupakan salah satu bentuk turunan dari arsitektur SDN secara garis besar. Model ini memungkin dalam sebuah jaringan untuk dapat memiliki lebih dari satu *controller* dan dapat saling berkomunikasi satu sama lain. Dalam model ini terdapat dua karakteristik yang berbeda, yaitu *logically centralized* dan *logically distributed*. *Logically centralized* merupakan sebuah karakteristik yang menerapkan konsep *multicontroller*, namun memusatkan titik kontrol jaringan pada salah satu *controller* saja. Sedangkan pada *logically distributed*, selain bentuk *controller* yang terdistribusi, beberapa *controller* dapat memiliki peran dan tanggung jawab yang sama sebagai pemegang kendali jaringan. Dalam implementasinya, model *distributed controller* juga bisa diterapkan sebagai *flat controller* ataupun *hierarchical controller* [3].

2.3. Open Flow Protocol

OpenFlow merupakan sebuah protokol komunikasi antara *control plane* dan *data plane*. OpenFlow telah dirancang untuk menyediakan aplikasi eksternal dengan akses ke forwarding plane pada *switch*. OpenFlow memiliki sebuah *secure channel* di antara OpenFlow *Switch* dan OpenFlow *Controller* yang memungkinkan adanya komunikasi dengan memberikan izin kepada *control plane* untuk mengirim instruksi, menerima permintaan, atau bertukar informasi [3].

2.4. Robustness

Permasalahan *robustness* pada penelitian ini mengacu pada permasalahan penempatan *controller* pada suatu jaringan SDN. Pada [13], dilakukan pengujian reliabilitas yang menggunakan pengujian *latency* pada beberapa model *controller*. Penelitian tersebut bertujuan untuk menemukan penempatan *controller* yang lebih optimal sesuai dengan kebutuhan tertentu.

2.5. Skalabilitas

Menurut [2], skalabilitas jaringan dapat dibagi menjadi dua, yaitu dengan melakukan penambahan perangkat jaringan dan menambah *traffic flow* pada suatu jaringan. Hal ini menjadi permasalahan yang mendasar pada model *single controller* karena pusat kontrol jaringan dan seluruh beban *traffic* dikelola secara terpusat oleh sebuah *controller*. Pada artikel tersebut diteliti dampak skalabilitas jaringan terhadap model *single controller* dengan menggunakan OpenDaylight dan ONOS sebagai *controller*. Pengujian dilakukan dengan penambahan jumlah host, switch, dan transmisi paket per detik. Hasil pengujian menunjukan bahwa pada saat keadaan jaringan belum mencapai titik jenuh setelah dilakukan skalabilitas jaringan, persentase tinkat kesuksesan paket diterima tetap konstan di 100%. *Delay* tetap stabil dengan nilai yang mendekati nol dan tingkat *throughput* berada di angka yang stabil saat dilakukan penambahan jumlah host dan switch. Namun, ketika keadaan jaringan sudah mencapai titik jenuh, nilai *throughput* mulai menurun, jumlah paket yang diterima di bawah persentase transimisi yang sukses dan *delay* meningkat secara signifikan.

2.6. Clustering

Metode ini merupakan proses penggabungan antara dua atau lebih *cluster* jaringan yang dikelola oleh *controller* yang berbeda. Hal ini bertujuan untuk meringankan beban *controller* pada model *single controller* dengan cara menambah *controller* lain pada jaringan tersebut dan menjadikan seluruh *controller* berada pada satu buah *cluster* jaringan yang sama. Peran *controller* sebagai pusat kontrol bisa menjadi *controller* utama ataupun *backup*. Tanpa adanya *clustering*, selain meningkatnya beban pada sebuah *controller*, permasalahan lain sangat mungkin terjadi ketika sebuah *controller* sudah mencapai titik jenuh. Jika suatu *controller* sudah dapat mencapai titik jenuh, *controller* tersebut dapat mengalami kegagalan perangkat yang berakibat fatal bagi jaringan yang dikelola [6].

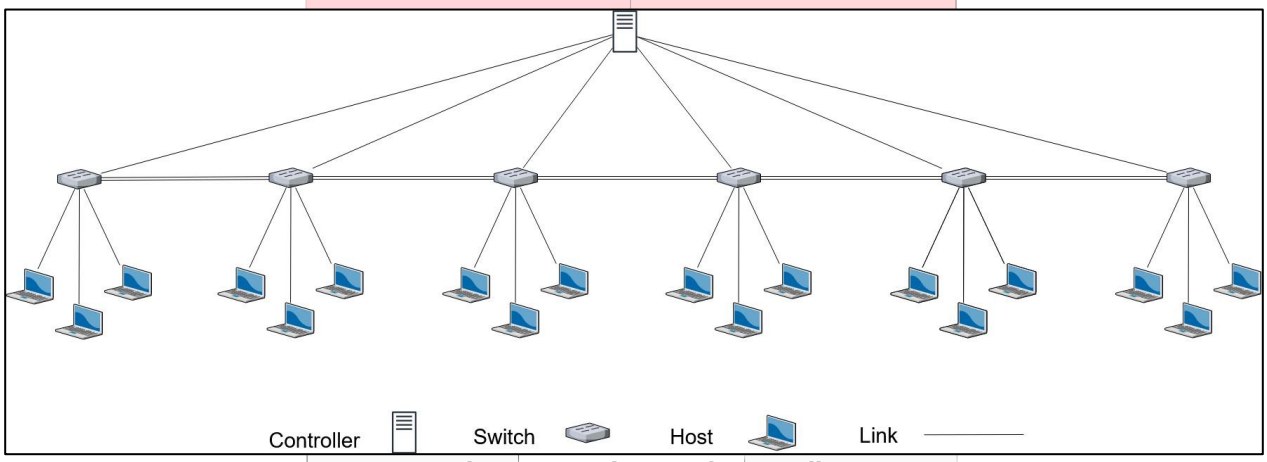
3. Sistem yang Dibangun

Penelitian ini diawali dengan perancangan topologi untuk membangun infrastruktur jaringan yang diuji. Kemudian dilanjut dengan perancangan *software* yang digunakan untuk mengimplementasikan pengujian pada penelitian ini berdasarkan analisis kebutuhan *software* yang mendukung proses pengujian. Terdapat dua topologi yang dirancang untuk penelitian ini, yaitu *single controller* dan *flat controller*.

Kedua topologi ini dirancang dengan penambahan jumlah switch dan host untuk mengetahui dampak dari skalabilitas jaringan terhadap kinerja jaringan dan penggunaan *resource* pada *controller*. Berikut tabel data jumlah switch dan host yang dirancang.

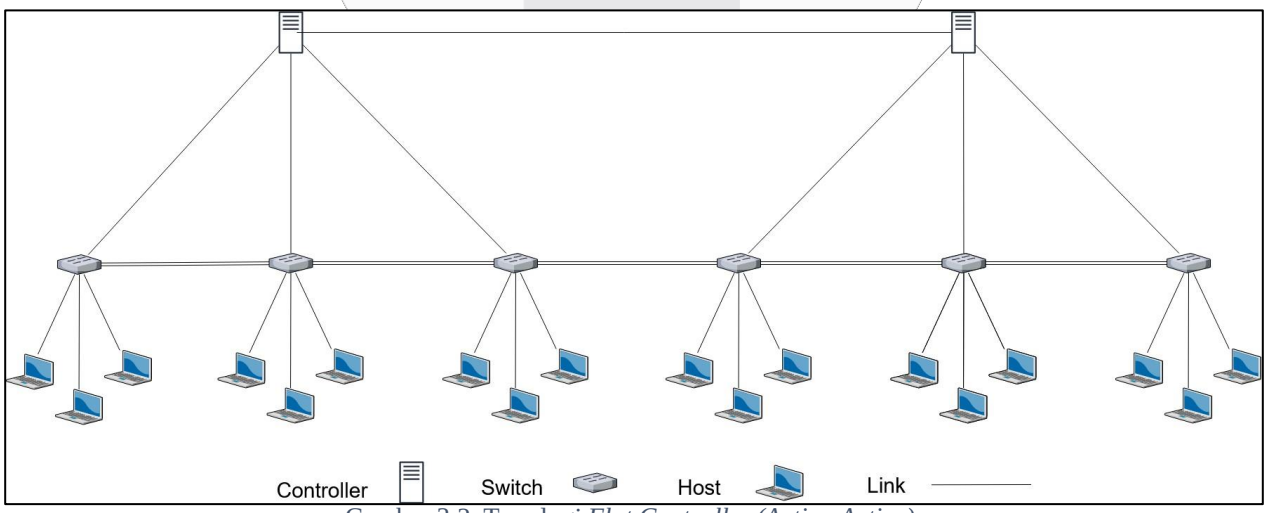
Tabel 3.1. Jumlah Node

No.	Switch	Host
1.	6	18
2.	12	36
3.	18	54
4.	24	72
5.	30	90
6.	36	108



Gambar 3.1. Topologi *Single Controller*

Perancangan topologi pada Gambar 3.1. dilakukan dengan menggunakan dua buah VM, yaitu VM1 berfungsi untuk menjalankan program mininet dan VM2 berfungsi untuk menjalankan *controller*. Pada topologi ini, seluruh switch terhubung pada *controller* tersebut.



Gambar 3.2. Topologi *Flat Controller (Active-Active)*

Perancangan topologi pada Gambar 3.2. dilakukan dengan menggunakan tiga buah VM. VM1 berfungsi untuk menjalankan program mininet, VM2 dan VM3 berfungsi untuk menjalankan *controller*. Penelitian tugas akhir ini merupakan pengembangan dari penelitian yang telah dilakukan sebelumnya pada [5] yang menggunakan *flat controller* dengan dua buah *controller* sebagai metode penelitian. Pada penelitian tersebut disebutkan bahwa *flat controller*(*active-active*) dapat meringankan beban kerja pada *controller* karena adanya dua *controller* yang sejajar dan aktif secara bersamaan. Oleh sebab itu, pada penelitian ini digunakan dua buah *controller* pada topologi *flat controller*(*active-active*) sebagai metode penelitian. Kedua controller tersebut memiliki peran yang sama, yaitu sebagai *controller* utama. Hal ini berutujuan memberikan akses yang sama kepada kedua controller terhadap jaringan yang dikelola dan juga untuk menyeimbangkan *load* pada kedua *controller*, sehingga penurunan kinerja jaringan dapat diminimalisasi. Perancangan topologi dibagi menjadi dua buah *cluster* berdasarkan jumlah switch yang terhubung pada *controller* dan menghasilkan jumlah switch yang seimbang di antara kedua *controller*. Seperti pada gambar di atas, setiap *controller* terhubung dengan tiga buah *switch*. Agar kedua *controller* dapat saling berkomunikasi dan terhubung dengan *switch* yang lainnya, proses *clustering* perlu dilakukan untuk menghubungkan dua buah *controller* tersebut. Terhubungnya kedua *controller* dan seluruh jaringan ditunjukkan dengan garis putus-putus pada kedua *controller*. Proses ini dilakukan dengan cara mengoperasikan *command* seperti pada gambar berikut dengan menambahkan IP address yang dimiliki oleh kedua *controller*.

```
dbvm3@dbvm3-VirtualBox:~$ sudo /opt/onos/bin/onos-form-cluster 10.0.2.5 10.0.2.6
Forming cluster on 10.0.2.5...
Forming cluster on 10.0.2.6...
```

Gambar 3.3. *Clustering* pada *Flat Controller*

Pada Gambar 3.3. dijalankan *command* onos-form-cluster diikuti dengan IP address dari *controller leader* atau *controller* utama dengan 10.0.2.5 dan IP address *controller follower* dengan 10.0.2.6. Setelah itu, kedua *controller* akan saling terhubung satu sama lain.

```
2021-01-08 10:05:35,791 | INFO  | rver-partition-0 | RaftContext
| 90 - io.atomix - 2.0.23 | RaftServer{partition-0} - Transitioning to FOLLOWER
2021-01-08 10:05:39,000 | INFO  | rver-partition-0 | RaftContext
| 90 - io.atomix - 2.0.23 | RaftServer{partition-0} - Found leader 10.0.2.6
```

Gambar 3.4. *Message log clustering* 1

Gambar di atas memperlihatkan proses pembentukan *controller* kedua sebagai FOLLOWER berdasarkan urutan IP address yang dimasukkan pada perintah onos-form-cluster di gambar 3.3. Pada gambar selanjutnya, *controller* pertama akan ditunjuk sebagai *leader* atau *controller* utama.

```
2021-01-08 10:06:18,263 | INFO  | rver-partition-0 | RaftContext
| 90 - io.atomix - 2.0.23 | RaftServer{partition-0} - Transitioning to LEADER
2021-01-08 10:06:18,274 | INFO  | rver-partition-0 | RaftContext
| 90 - io.atomix - 2.0.23 | RaftServer{partition-0} - Found leader 10.0.2.5
```

Gambar 3.5 *Message log clustering* 2

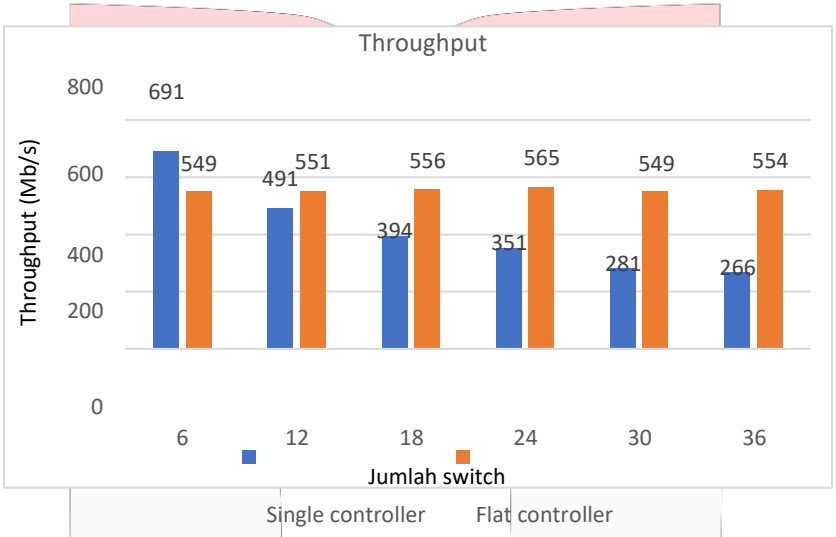
Pada gambar 3.5., *controller* pertama dengan IP address 10.0.2.5 diberikan peran sebagai LEADER atau *controller* utama berdasarkan urutan IP address pada perintah pembentukan *clustering* seperti pada gambar 3.3. Dalam pelaksanaan perancangan topologi dibutuhkan RAM pada masing-masing VM sebesar 2GB dengan *storage* sebesar 20GB dan menggunakan dua CPU. Beberapa *software* lainnya yang dibutuhkan adalah sistem operasi Ubuntu 18.04.4 LTS dan python yang digunakan untuk merancang topologi pada mininet.

4. Evaluasi

Pada bagian ini akan dijelaskan pengujian yang dilakukan dan hasil dari pengujian tersebut. Pengujian yang dilakukan pada penelitian ini, yaitu pengujian kinerja jaringan, penggunaan *resource* pada *controller*, dan reliabilitas. *Throughput* dan *latency* menjadi parameter pengujian kinerja jaringan. Sedangkan penggunaan CPU dan *memory* menjadi parameter pengujian pengujian penggunaan *resource* pada *controller*. Pada pengujian reliabilitas dilakukan pengujian *latency* seperti pada pengujian kinerja jaringan, namun dilakukan panambahan *switch*.

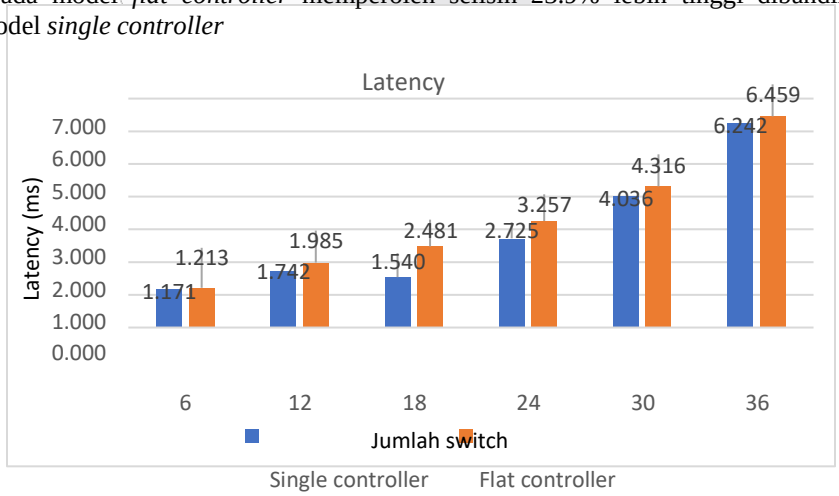
4.1. Pengujian Kinerja Jaringan

Pengujian ini bertujuan untuk mengetahui dampak skalabilitas jaringan terhadap kinerja jaringan pada model *single controller* dan *flat controller*. Simulasi pada pengujian ini dibagi dua berdasarkan parameter kinerja jaringan, yaitu *throughput* dan *latency*. Pengujian *throughput* dilakukan pada terminal *xterm* dengan menggunakan perintah *iperf* untuk mengirimkan paket TCP yang dilakukan sebanyak 10 kali percobaan, kemudian didapat nilai rata-rata dari keseluruhan percobaan. Selain itu juga terdapat pengujian *latency* yang dilakukan pada terminal *xterm* dengan menggunakan perintah *ping* dari *host* 1 ke *host* terjauh untuk mengukur lama waktu (RTT) pengiriman paket dari satu host ke host lainnya sampai paket tersebut diterima kembali oleh host pertama. Penguajian ini dilakukan dengan mengambil pengujian pertama dimana terdapat *delay* dari paket yang dikirimkan ke *controller* terlebih dahulu sebelum diteruskan ke *switch* dan *host* tujuan. Simulasi pengujian *throughput* dilakukan sebanyak 10 kali percobaan dengan setiap percobaan dilakukan selama 10 detik dengan pengaturan *bandwidth* sebesar satu Gigabit per second (Gb/s). Pengujian *latency* dilakukan sebanyak 10 kali percobaan. Dalam setiap percobaan terdapat pengiriman 10 paket dan didapat nilai rata-ratanya. Kemudian dari rata-rata tersebut didapatkan kembali nilai rata-rata dari 10 percobaan.



Gambar 4.1. Pengujian *Throughput*

Berdasarkan hasil pengujian pada gambar 4.1, didapatkan nilai *throughput* pada model *single controller* dengan jumlah *switch* sebanyak 6-36 buah mengalami penurunan angka dari 691-266 Mb/s dengan nilai rata-rata 421.53 Mb/s. Sedangkan nilai *throughput* pada model *flat controller* dengan penambahan jumlah *switch* yang sama mendapatkan angka yang relatif stabil pada kisaran 549-565 Mb/s dengan nilai rata-rata 553.85 Mb/s. Berdasarkan hasil tersebut, nilai *throughput* pada model *flat controller* memperoleh selisih 23.9% lebih tinggi dibandingkan dengan nilai *throughput* pada model *single controller*



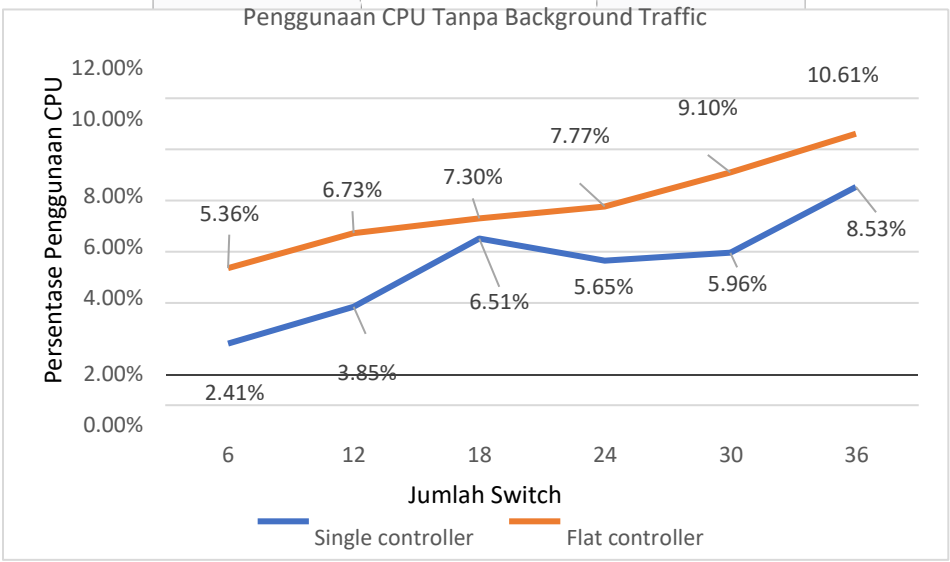
Gambar 4.2. Pengujian *Latency*

Berdasarkan grafik pada gambar 4.2. diperoleh hasil pengujian *latency* pada model *single controller* berkisar 1.171-6.242 ms dengan nilai rata-rata 2.909 ms. Sedangkan nilai *latency* pada pada model *flat controller* berkisar 1.213-6.459 ms dengan nilai rata-rata 3.285 ms. Berdasarkan hasil tersebut, nilai *latency* pada model *single controller* memperoleh selisih 11.45% lebih rendah dibandingkan dengan nilai *latency* pada model *flat controller*. Pada gambar tersebut dapat terlihat, baik model *single controller* maupun *flat controller* memperoleh nilai *latency* yang meningkat.

Berdasarkan pengujian *throughput* pada kedua model *controller*, diperoleh hasil yang tidak mencapai kapasitas maksimum *bandwidth* sebesar 1 Gbps. Hal ini disebabkan oleh bentuk topologi linear pada kedua model *controller*. Banyaknya *link* antar *switch* yang dilalui sebuah paket untuk sampai ke tujuan mengakibatkan tidak tercapainya nilai maksimum dari *throughput* pada kedua model *controller*. Selain itu, pada pengujian *latency* didapatkan nilai yang meningkat pada kedua model *controller*. Nilai rata-rata *latency* yang meningkat dimulai dari 1 ms disebabkan oleh *reactive forwarding* untuk metode penerusan paket pada *switch*. Metode ini bekerja dengan cara mengirimkan paket yang pertama kali ke *controller* sebelum diteruskan ke tujuan, sehingga menghasilkan *delay* tambahan dari *controller* untuk proses pengiriman paket tersebut. Nilai *delay* akan menurun setelah paket pertama karena *switch* sudah bisa meneruskan paket sendiri tanpa harus meminta konfirmasi pengiriman paket *controller* lagi.

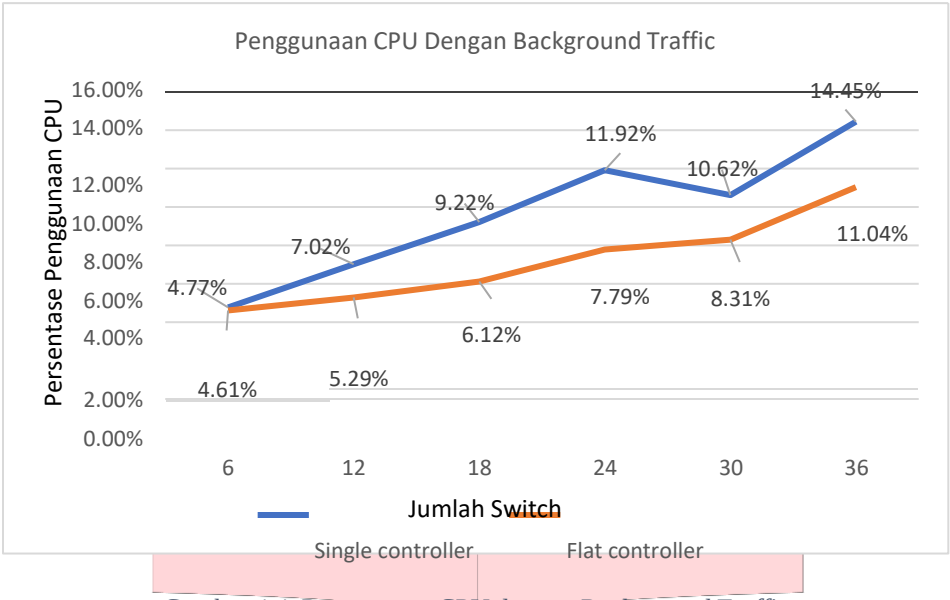
4.2. Pengujian Penggunaan CPU dan *Memory*

Pengujian penggunaan CPU dan *memory* bertujuan untuk mengetahui pengaruh dari skalabilitas jaringan terhadap penggunaan *resource* pada *controller* dengan model *single controller* dan *flat controller(active-active)*. Pengujian ini dilakukan tanpa *background traffic* dan dengan *background traffic*. Selain untuk mengetahui pengaruh dari skalabilitas jaringan, pada pengujian ini juga terdapat skenario lainnya untuk melihat pengaruh dari penambahan jumlah *traffic* pada penggunaan CPU dan memory. Seluruh percobaan pengujian dilakukan sebanyak 10 kali saat proses iperf berlangsung. Setiap percobaan pengujian dilakukan dalam satu interval selama 10 detik, setelah itu dapat diambil data penggunaan CPU dan *memory*. Dalam pengujian penambahan *traffic*, proses iperf dilakukan dengan menggunakan enam *host* pada enam *switch* yang berbeda. Dimana tiga *host* pertama sebagai *server* dan tiga *host* lainnya sebagai *client*. pada kedua model *controller* dimana satu percobaan dilakukan dengan.



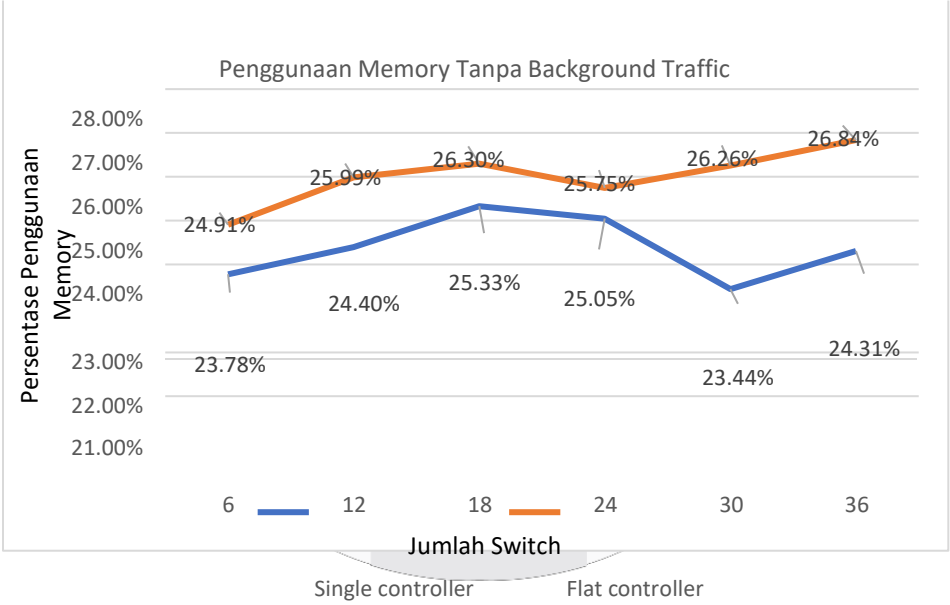
Gambar 4.3. Penggunaan CPU Tanpa *Background Traffic*

Hasil pengujian penggunaan CPU tanpa *background traffic* pada gambar 4.3. menunjukkan bahwa penggunaan CPU pada kedua model *controller* meningkat dimulai dari jumlah *switch* sebanyak 6 buah hingga 36 buah. Model *flat controller* menghasilkan nilai tertinggi sebesar 10.61%. Pada model *single controller* terlihat penggunaan CPU tertinggi sebesar 8.53%. Berdasarkan hasil tersebut, model *single controller* mendapatkan selisih 2.08% lebih rendah dibandingkan dengan model *flat controller*.



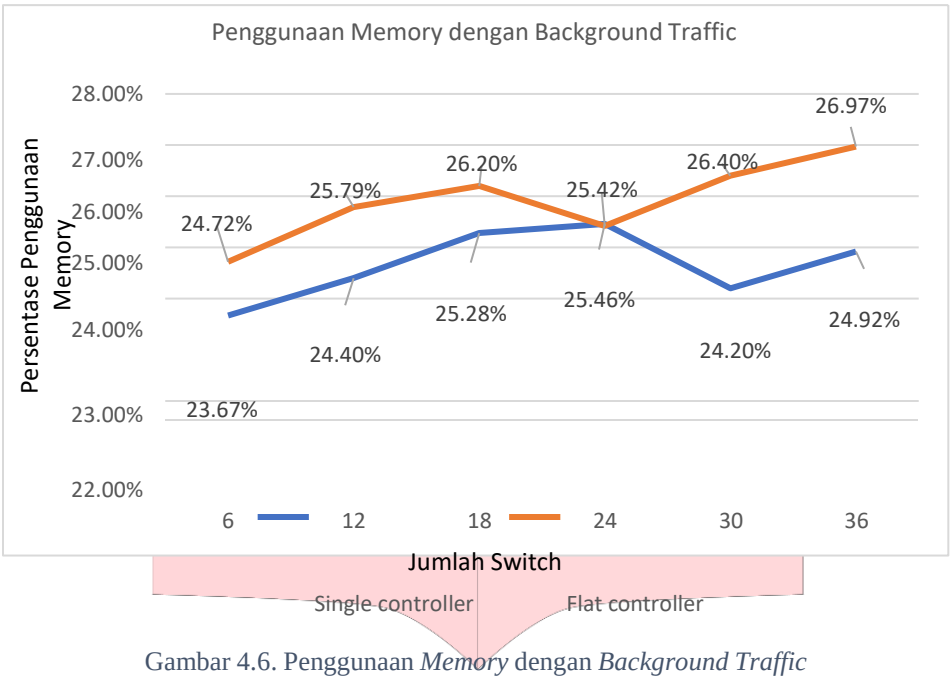
Gambar 4.4. Penggunaan CPU dengan *Background Traffic*

Grafik pada gambar 4.4. menunjukkan bahwa model *flat controller* menghasilkan penggunaan CPU yang lebih rendah dibandingkan dengan model *single controller*. Nilai persentase tertinggi dihasilkan oleh model *single controller* sebesar 14.45%, sedangkan model *flat controller* mendapatkan nilai tertinggi sebesar 11.04%. Berdasarkan hasil tersebut, model *flat controller* mendapatkan selisih 3.41% lebih rendah dibandingkan dengan model *single controller*.



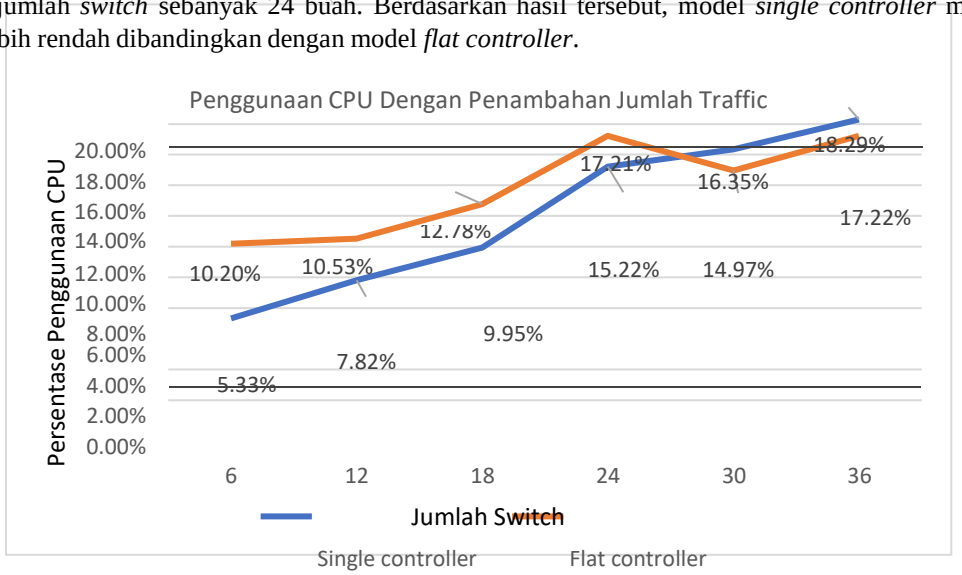
Gambar 4.5. Penggunaan *Memory* tanpa *Background Traffic*

Hasil pengujian pada penggunaan *memory* tanpa *background traffic* pada gambar 4.5. menunjukkan bahwa kedua model *controller* memperoleh nilai penggunaan *memory* yang meningkat, namun fluktuatif. Nilai persentase tertinggi yang didapatkan oleh model *flat controller* adalah 26.84% saat jumlah *switch* sebanyak 36 buah. Sedangkan, nilai persentase tertinggi yang didapatkan oleh model *single controller* adalah 25.33% saat jumlah *switch* sebanyak 18 buah. Berdasarkan hasil tersebut, model *single controller* memperoleh selisih 1.51% lebih rendah dibandingkan dengan model *flat controller*.



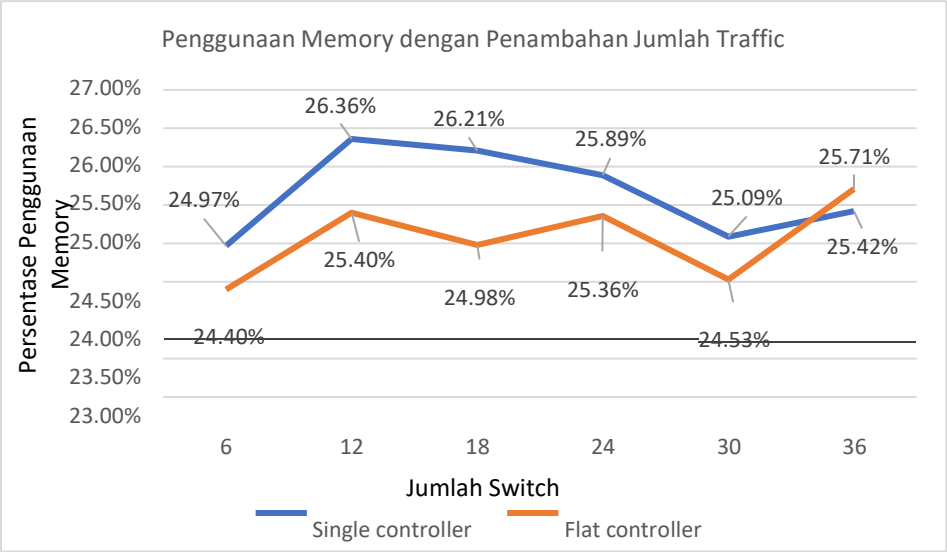
Gambar 4.6. Penggunaan *Memory* dengan *Background Traffic*

Pada gambar 4.6. terlihat grafik yang menyerupai pada gambar 4.5. Gambar ini menunjukkan hasil pengujian penggunaan *memory* dengan *background traffic*, nilai persentase tertinggi pada model *flat controller* sebesar 26.97% ketika jumlah *switch* sebanyak 36 buah. Sedangkan, nilai persentase tertinggi pada model *single controller* sebesar 25.46% ketika jumlah *switch* sebanyak 24 buah. Berdasarkan hasil tersebut, model *single controller* memperoleh selisih 1.51% lebih rendah dibandingkan dengan model *flat controller*.



Gambar 4.7. Penggunaan CPU dengan Penambahan Jumlah *Traffic*

Grafik pada gambar 4.7. menunjukkan bahwa model *flat controller* menghasilkan penggunaan CPU yang lebih tinggi dari model *single controller* pada saat jumlah *switch* sebanyak enam sampai dengan 24 buah. Namun, terdapat penurunan persentase penggunaan CPU saat jumlah *switch* sebanyak 30 buah. Pada jumlah *switch* sebanyak 36 buah, model *flat controller* memiliki persentase penggunaan CPU yang lebih rendah dibandingkan dengan model *single controller*. Nilai persentase tertinggi dihasilkan oleh model *single controller* sebesar 18.29%, sedangkan model *flat controller* mendapatkan nilai tertinggi sebesar 17.22%. Berdasarkan hasil tersebut, model *flat controller* mendapatkan selisih 1.07% lebih rendah dibandingkan dengan model *single controller*.



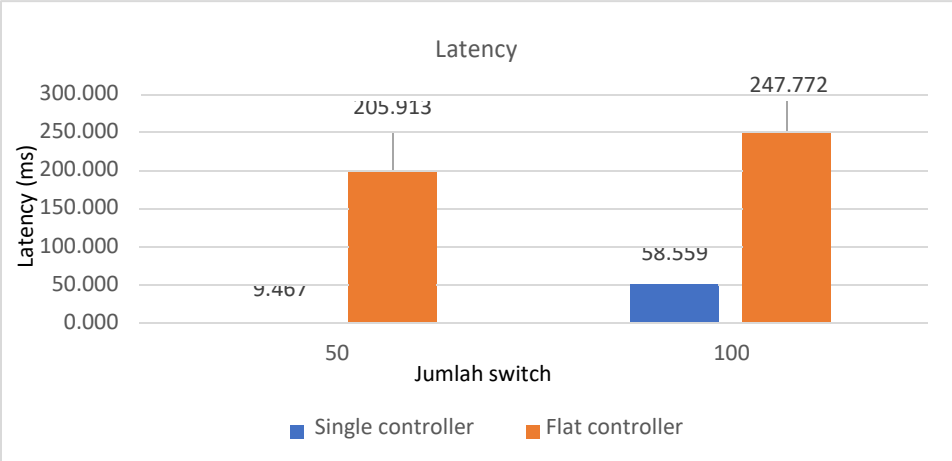
Gambar 4.8. Penggunaan *Memory* dengan Penambahan Jumlah *Traffic*

Pada gambar 4.8. ditunjukkan hasil penggunaan *memory* dengan penambahan jumlah *traffic*. Model *flat controller* menghasilkan penggunaan *memory* yang lebih rendah pada jumlah *switch* sebanyak 6-30 buah. Namun, terdapat peningkatan persentase penggunaan *memory* saat jumlah *switch* sebanyak 36 buah. Secara keseluruhan, penggunaan *memory* pada model *single controller* menghasilkan nilai yang lebih besar dibandingkan dengan model *flat controller*. Model *single controller* memperoleh persentase tertinggi pada jumlah *switch* sebanyak 12 buah dengan nilai 26.36%. Sedangkan model *flat controller* memperoleh nilai tertinggi pada saat jumlah *switch* sebanyak 36 buah dengan nilai 25.71%.

Berdasarkan seluruh hasil pengujian CPU dan *memory*, baik model *single controller* maupun *flat controller* menghasilkan nilai yang meningkat seiring dengan bertambahnya jumlah *switch* dan *traffic*. Dengan kata lain, semakin banyak jumlah *switch* dan *traffic* yang digunakan akan menghasilkan penggunaan CPU dan *memory* yang semakin besar. Penggunaan *memory* pada model *flat controller* menghasilkan nilai yang lebih tinggi dibandingkan dengan model *single controller* pada hasil pengujian yang menggunakan *background traffic* dan tanpa *background traffic*. Hal ini disebabkan oleh terhubungnya kedua *controller*, sehingga terdapat penggunaan *memory* yang lebih besar. Namun, dengan terhubungya kedua *controller* tersebut juga dapat menurunkan nilai penggunaan CPU ketika terdapat *background traffic* dan jumlah *traffic* ditingkatkan. Hal ini disebabkan oleh adanya dua buah *controller* yang bekerja secara bersamaan, sehingga dapat menurunkan beban kerja dengan cara membagi beban kerja tersebut ke dua buah *controller*. Jika dibandingkan dengan model *single controller*, beban kerja *controller* pada model ini hanya diproses oleh satu *controller* saja, sehingga menghasilkan penggunaan CPU yang lebih besar. Berdasarkan pengujian tersebut, model *flat controller* menghasilkan penggunaan *resource* pada *controller* yang lebih baik dibandingkan dengan model *single controller*.

4.3. Pengujian Reliabilitas (Latency)

Pengujian reliabilitas bertujuan untuk mengetahui model *controller* mana yang memiliki reliabilitas lebih baik berdasarkan pengujian *latency*. Langkah yang dilakukan pada pengujian ini hampir sama dengan pengujian *latency* pada pengujian kinerja jaringan, yang membedakan hanya pada jumlah *switch* yang digunakan. Pada pengujian reliabilitas digunakan 50 dan 100 buah *switch*. Data *latency* yang diambil juga sama dengan pengujian sebelumnya, yaitu dilakukan pengambilan data pertama yang akan menghasilkan nilai *delay* tambahan dari proses pada *controller*.



Gambar 4.9. Pengujian Reliabilitas (*Latency*)

Hasil pengujian pada gambar 4.9. menunjukkan nilai *latency* dari penambahan jumlah *switch* pada kedua model *controller*. Model *flat controller* memperoleh *latency* tertinggi dengan nilai 247.772 ms, sedangkan nilai tertinggi pada model *single controller* adalah 58.559 ms. Model *flat controller* menghasilkan nilai *latency* yang lebih tinggi karena terdapat *controller* tambahan yang perlu melakukan pemrosesan paket pertama setelah perintah ping dijalankan. Walaupun pada paket kedua dan seterusnya nilai *latency* akan turun kembali karena setiap *switch* sudah mengetahui tujuan paket tersebut. Jika dibandingkan dengan model *single controller*, *controller* yang digunakan hanya terdapat satu buah saja, sehingga nilai *delay* tambahan hanya dihasilkan oleh satu buah *controller* controller saja.

5. Kesimpulan

Berdasarkan pengujian kinerja jaringan, serta penggunaan CPU dan *memory* dengan melakukan perbandingan kinerja antara model *single controller* dan *flat controller(active-active)*, diketahui model *flat controller(active-active)* dengan parameter *throughput* mendapatkan hasil yang lebih tinggi dan stabil, serta *latency* yang lebih rendah dibandingkan dengan model *single controller*. Pada pengujian penggunaan CPU dengan *background traffic* dan penambahan jumlah *traffic*, model *flat controller(active-active)* menghasilkan penurunan nilai pada jumlah *switch* tertentu dibandingkan dengan model *single controller*. Hal ini disebabkan oleh aktifnya kedua *controller* secara bersamaan yang terjadi setelah diterapkan metode *clustering*, sehingga terdapat penggunaan *resource* yang lebih rendah dibandingkan dengan model *single controller*. Penggunaan *resource* yang lebih besar pada model *single controller* disebabkan oleh beban kerja atau *load* pada *controller* yang diproses oleh satu buah *controller* saja.

Pada pengujian reliabilitas didapatkan hasil bahwa model *flat controller* menghasil nilai *latency* yang lebih besar dibandingkan dengan model *single controller*. Hal ini disebabkan oleh adanya *controller* tambahan yang menghasilkan *delay* tambahan pada proses pengiriman paket pertama pada jaringan tersebubut. Sedangkan, model *single controller* hanya menggunakan satu *controller*, sehingga nilai *delay* tamabahan hanya dihasilkan oleh satu *controller* tersebut saja. Berdasarkan pengujian kinerja jaringan dan penggunaa *resource* pada *controller* yang telah dilakukan, skalabilitas jaringan mempengaruhi kinerja jaringan, serta penggunaan CPU dan *memory* baik pada model *single controller* maupun *flat controller*. Berdasarkan pengujian reliabilitas, dapat disimpulkan bahwa dibandingkan dengan model *flat controller(active-active)*, model *single controller* memiliki *robustness* terhadap skalabilitas jaringan yang lebih baik dibandingkan dengan model *single controller*. Saran untuk penelitian selanjutnya dapat menggunakan model *hierarchical controller* sebagai metode penelitian.

Referensi

[1] E. Amiri, E. Alizadeh, and K. Raeisi, “An Efficient Hierarchical Distributed SDN Controller Model,” in *2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI)*, 2019, pp. 553–557.

[2] S. Chaipet and W. Putthividhya, “On Studying of Scalability in Single-Controller Software-Defined Networks,” *2019 11th Int. Conf. Knowl. Smart Technol. KST 2019*, pp. 158–163, 2019.

[3] O. Blial, M. Ben Mamoun, and R. Benaini, “An Overview on SDN Architectures with Multiple Controllers,” vol. 2016, 2016.

[4] T. Hu, Z. Guo, P. Yi, T. Baker, and J. Lan, “Multi-controller Based Software-Defined Networking: A

Survey," *IEEE Access*, vol. 6, no. c, pp. 15980–15996, 2018.
[5] T. Akhir, "Desain Distributed Controller dengan Metode Active-Active pada Jaringan Software Define Network Program Studi Sarjana Informatika Fakultas Informatika Universitas Telkom Bandung," 2019.
[6] A. Abdelaziz *et al.*, "Distributed controller clustering in software defined networks," *PLoS One*, vol. 12, no. 4, pp. 1–19, 2017.
[7] Y. Jimenez, J. A. Cordero, and C. Cervello-Pastor, "Measuring robustness of SDN control layers," *Proc. 2015 IFIP/IEEE Int. Symp. Integr. Netw. Manag. IM 2015*, pp. 774–777, 2015.
[8] L. Mamushiane, A. Lysko, and S. Dlamini, "A comparative evaluation of the performance of popular SDN controllers," *IFIP Wirel. Days*, vol. 2018-April, pp. 54–59, 2018.
[9] N. A. Faruqi, L. Nurwadi, N. Ismail, and D. Maryanto, "Simulasi Kinerja Berbagai Topologi Jaringan Berbasis Software-Defined Network (SDN)," *Senter*, vol. 3, pp. 232–239, 2017.
[10] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, 2015.
[11] M. Paliwal, D. Shrimankar, and O. Tembhurne, "Controllers in SDN : A Review Report," *IEEE Access*, vol. PP, no. March 2011, p. 1, 2018.
[12] Y. E. Oktian, S. Lee, H. Lee, and J. Lam, "Distributed SDN controller system : A survey on design choice," vol. 121, pp. 100–111, 2017.
[13] F. Bannour, S. Souihi, and A. Mellouk, "Scalability and reliability aware SDN controller placement strategies," in *2017 13th International Conference on Network and Service Management (CNSM)*, 2017, vol. 2018-Janua, pp. 1–4.