IMPLEMENTASI *LOAD BALANCING* UNTUK KONTROLER *SOFTWARE DEFINED NETWORK*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Adi Iman Utama
NIM: 135150201111098

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

IMPLEMENTASI *LOAD BALANCING* UNTUK KONTROLER *SOFTWARE DEFINED NETWORK*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Adi Iman Utama
NIM: 135150201111098

Skripsi ini telah diuji dan dinyatakan lulus pada
27 April 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Widhi Yahya, S.Kom., M.Sc	Dany Primanita Kartikasari, S.T., M.Kom
NIK: 2016078911211001	NIP: 197711162005012003

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 19710518 200312 1 001

Identitas Tim Penguji

1. Fariz Andri Bakhtiar, S.T., M.Kom (NIK. 2017098403141001)

2. Rakhmadhany Primananda, S.T, M.Kom (NIK. 2016098604061001)

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 27 April 2018

Adi Iman Utama

NIM: 135150201111098

Daftar Riwayat Hidup

Nama	**: Adi Iman Utama**
Tempat, tanggal lahir	**: TulungAgung, 19 Agustus 1994**
Jenis Kelamin	**: Laki – Laki**
Agama	**: Islam**
Alamat	**: Perumahan Graha Kuncara M-23 Sidoarjo**
No Tlp	**: 082234218553**

Latar Belakang Pendidikan
2004 – 2007 SDN Pucang 1 Sidoarjo
2007 – 2010 SMPN 5 Sidoarjo
2010 – 2013 SMAN 4 Sidoarjo
2013 – 2018 Universitas Brawijaya Fakultas Ilmu Komputer

Terimakasih Kepada

1. Bapak Widhi Yahya, S.Kom., M.Sc. selaku dosen pembimbing I yang dalam penulisan ini telah banyak memberi ilmu serta masukan.
2. Ibu Dany Primanita Kartikasari, S.T., M.Kom. selaku dosen pembimbing II yang juga banyak memberi ilmu dan masukan untuk penulisan laporan skripsi ini.
3. Keluarga penulis, khususnya yaitu orang tua penulis Bapak Saksono Banu Wasito dan Ibu Heni Nur Hayati, serta adik penulis yaitu Annisa Cahya Fatika yang telah memberikan dukungan moril dan materil serta motivasi.
4. Hudan Abdur R, Teguh Surya, Yogi Suwandy, Malik Abdul Azis, Jefry Kalvin, Artiyan Prasetya, Bayu Laksana Yudha, M. Alfi Fauzan, Fransnesa, dan Dhyono Dhyakso, selaku teman serta sahabat yang selalu membantu dan memberikan motivasi selama pengerjaan skripsi ini.
5. Teman-teman Fakultas Ilmu Komputer, khususnya teman-teman program studi Informatika 2013 terima kasih atas segala bantuan dan dukungannya selama ini.
6. Teman-teman Komunitas Ketimbang Ngemis Malang yang selalu memberikan dukungan moril selama pengerjaan skripsi ini.
7. Seluruh pihak yang telah membantu secara langsung dan tidak langsung dalam penyelesaian tugas akhir ini yang namanya tidak dapat disebutkan satu per satu oleh penulis.

ABSTRAK

Konsep jaringan tradisional dalam beberapa dekade ini pada umumnya dirasa sudah cukup mengesankan, tetapi dalam perkembangannya dilihat dari segi ukuran dan kompleksitas jaringan membuat pengoperasikan peralatan dan kebutuhan untuk mempercepat inovasi jaringan semakin meningkat. Sehingga dibutuhkan solusi untuk perkembangan dari kebutuhan jaringan yang semakin kompleks yaitu dengan penekanan biaya untuk investasi perangkat keras jaringan dengan menggunakan *Software Defined Network (SDN).* SDN merupakan sebuah konsep pemisahan antara *control plane* dan *data plane* pada perangkat jaringan seperti *router* dan *switch,* dimana pada jaringan tradisional *control plane* dan *data plane* ini digabung menjadi satu. Dengan menggunakan konsep SDN ini juga masih mengalami penurunan performa ketika jumlah *switch* yang terhubung dalam lingkungan jaringan bertambah ketika digunakan *single* kontroler, maka dari itu dalam penelitian ini digunakan *load balancing* untuk melakukan pembagian beban pada beberapa kontroler yang terhubung. Kontroler yang digunakan dalam penelitian ini adalah POX sedangkan untuk mengimplementasikan *load balancing* kontroler akan digunakan HAproxy pada komputer server. Hasil pengujian menunjukkan bahwa seiring bertambahnya jumlah kontroler dengan diberikan *load balancing* akan memberikan hasil *latency* dan *throughput* yang semakin baik, tetapi seiring bertambahnya jumlah kontroler maka beban dari server *load balancing* akan semakin meningkat dilihat dari CPU dan Memori *Usage*.

Kata Kunci : *Software Defined Network (SDN)*, *Load balancing*, *HAproxy,* kontroler

ABSTRACT

The concept of traditional networks in decades is generally considered quite impressive, but in it's development in terms of size and complexity of the network make the equipment operational and the need to accelerate network innovation is increasing. So it takes a solution to the development of the increasingly complex network needs that is with the emphasis of the cost for investment in network hardware using Software Defined Network (SDN). SDN is a concept of separation between control plane and data plane on network devices such as routers and switches, in which traditional network control plane and data plane are combined into one. By using the concept of SDN is also still decreased performane when the number of switches connected in a network environtment increases when used single controller, therefore in this research used load balancing to do load sharing on some connected controller. The controller used in this research is POX while to implement load balancing controller will be used HAproxy on server computer. The test results show that as the increasing number of controllers with load balancing will result in better latency and throughput, but as the number of controllers increases load of server load balancing will be seen from CPU dan Memory usage.

Keywords: Software Defined Network (SDN), Load balancing, HAproxy, controller

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah SWT, karena berkat rahmat serta bimbingannya penulis dapat menyelesaikan tugas akhir dengan judul "IMPLEMENTASI *LOAD BALANCING* UNTUK *KONTROLER SOFTWARE DEFINED NETWORK*" dengan baik dan tepat waktu. Penulisan dan penyusunan laporan skripsi ini dapat terlaksana dengan baik karena adanya bantuan secara langsung maupun tidak langsung dari pihak tertentu diantaranya:

1. Bapak Widhi Yahya, S.Kom., M.Sc. selaku dosen pembimbing I yang dalam penulisan ini telah banyak memberi ilmu serta masukan.
2. Ibu Dany Primanita Kartikasari, S.T., M.Kom. selaku dosen pembimbing II yang juga banyak memberi ilmu dan masukan untuk penulisan laporan skripsi ini.
3. Keluarga penulis, khususnya yaitu orang tua penulis Bapak Saksono Banu Wasito dan Ibu Heni Nur Hayati, serta adik penulis yaitu Annisa Cahya Fatika yang telah memberikan dukungan moril dan materil serta motivasi.
4. Hudan Abdur R, Teguh Surya, Yogi Suwandy, Malik Abdul Azis, Jefry Kalvin, Artiyan Prasetya, Bayu Laksana Yudha, M. Alfi Fauzan, Fransnesa, dan Dhyono Dhyakso, selaku teman serta sahabat yang selalu membantu dan memberikan motivasi selama pengerjaan skripsi ini.
5. Teman-teman Fakultas Ilmu Komputer, khususnya teman-teman program studi Informatika 2013 terima kasih atas segala bantuan dan dukungannya selama ini.
6. Teman-teman Komunitas Ketimbang Ngemis Malang yang selalu memberikan dukungan moril selama pengerjaan skripsi ini.
7. Seluruh pihak yang telah membantu secara langsung dan tidak langsung dalam penyelesaian tugas akhir ini yang namanya tidak dapat disebutkan satu per satu oleh penulis.

Semoga jasa dan amal baik mendapatkan balasan dari Allah SWT. Dengan segala kerendahan hati, penulis menyadari sepenuhnya bahwa skripsi ini masih jauh dari sempurna karena keterbatasan materi dan pengetahuan yang dimiliki penulis. Akhirnya semoga skripsi ini dapat bermanfaat dan berguna bagi pembaca terutama mahasiswa Filkom Universitas Brawijaya

Malang, 27 April 2018

Penulis

adimanutama@yahoo.com

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar belakang

Penggunaan perangkat jaringan dalam beberapa dekade ini seperti *router* dan *switch* telah digunakan dalam banyak lingkungan dan dapat menjalankan fungsi jaringan tradisional yaitu *filtering* dan *forwarding* paket menuju tujuan akhir. Walaupun perangkat jaringan ini sudah mengesankan, banyaknya perkembangan baik dalam segi ukuran dan kompleksitas membuat jaringan tradisional ini berkurang performanya dalam mengelola lalu lintas data (Goransson, P. dan Black, C., 2014). Alasan untuk fakta diatas mencakup biaya yang semakin meningkat untuk memiliki dan mengoperasikan peralatan jaringan, kebutuhan untuk mempercepat inovasi jaringan dan peningkatan permintaan pusat data modern (Goransson, P. dan Black, C., 2014). Solusi untuk perkembangan dari kebutuhan jaringan yang semakin kompleks yaitu dengan penekanan biaya untuk investasi perangkat keras jaringan. Inovasi yang paling tepat digunakan adalah *Software Defined Network* (SDN).

Konsep *Software Defined Network* ini melakukan pemisahan antara *control plane* dan *data plane* pada perangkat jaringan seperti *router* dan *switch*. *Control plane* adalah bagian yang berfungsi untuk mengatur logika jaringan seperti algoritma *routing* dan tabel *routing*, sedangkan *data plane* adalah bagian yang berfungsi untuk mengatur bagaimana paket akan diteruskan menuju *next hop* berikutnya setelah paket tiba pada *forwarding engine* (Chao, H.J., dan Bin, L., 2007) sedangkan pada jaringan umum biasanya *control plane* dan *data plane* ini tergabung dalam satu perangkat. Pada SDN kebutuhan performa yang baik untuk kontroler sangat diperlukan, melihat kompleksitas jaringan yang semakin meningkat mengakibatkan beban kontroler pun semakin bertambah. Kompleksitas jaringan terjadi ketika jumlah switch dalam jaringan semakin meningkat sehingga dalam kondisi tersebut penggunakan satu kontroler bukanlah pilihan yang tepat karena akan menghasilkan nilai *latency* yang tinggi untuk beberapa *flows*, hal tersebut disebabkan karena beban kerja yang tinggi pada satu kontroler yang tersedia (Vinayagamurthy, D., Balasundaram, J., 2012).

Dilihat dari permasalahan tersebut, maka penambahan jumlah kontroler dengan melakukan mekanisme *load balancing* dapat diterapkan untuk membagi rata beban kerja yang akan ditangani oleh masing-masing kontroler. Penerapan *load balancing* sangat membantu dalam permasalahan ini, dimana tujuan *load balancing* adalah memungkinkan pembagian beban menjadi seimbang (Guo, Z., Su, M., dan Duan, Z., 2014). Dengan menggunakan *load balancing* maka kinerja dari masing-masing kontroler akan menjadi seimbang dan *load balancing* ini dapat digunakan sebagai *backup* (*high availability*) jika ada salah satu kontroler yang mengalami kerusakan maka kontroler lainnya masih bisa melayani *traffic* jaringan yang sedang berjalan. *Load balancing* juga sudah banyak dikembangkan pada SDN, contohnya pada *web server* yang bertujuan untuk meringankan beban masing-masing *web server* dari *request client* tetapi masih menggunakan *single* kontroler.

Parameter keberhasilan dari *load balancing* kontroler pada SDN dapat dilihat berdasarkan *latency* dan *throughput* yang diberikan dari masing-masing kontroler. Dikatakan berhasil apabila nilai *latency* dan *throughput* dari penggunaan *load balancing* pada SDN ini lebih baik daripada *single* kontroler, namun parameter ini juga harus dilihat dari hasil CPU dan Memori *usage* pada server *load balancing* agar dapat diketahui juga seberapa besar beban yang diterima pada sisi server *load balancing*. Maka dari itu pada penelitian ini, performa dari server *load balancing* diukur berdasarkan parameter CPU dan Memori *usage*. Sedangkan performa dari tiap kontroler dapat diukur dengan menggunakanan parameter *flow setup rate* dan *flow setup delay*. *Flow setup rate* adalah banyak flow yang mampu direspon oleh kontroler, sedangkan *flow setup delay* adalah waktu yang dibutuhkan sebuah kontroler dalam memberikan respon (Vengainathan, B ., Anton, B., dan Vishwas, M., 2014).

Berdasarkan paparan di atas judul yang diambil dalam skripsi ini adalah "Implementasi *Load Balancing* Untuk Kontroler *Software Defined Network*". Kontroler yang digunakan pada penelitian ini adalah POX. POX adalah *platform open-source* yang menggunakan bahasa pemrograman *Python* serta memungkinkan proses perancangan dan pembangunan jaringan lebih cepat dan lebih umum dibandingkan pendahulunya NOX (Al-Shabibi, A., dan McCauley, M., 2015). Fokus dari penelitian ini adalah untuk mengetahui efek penerapan *load balancing* kontroler dalam menangani sebuah *traffic* pada jaringan dengan melakukan pengujian menggunakan parameter uji *flow setup rate* dan *flow setup delay* pada kontroler.

1.2 Rumusan masalah

Berdasarkan latar belakang yang telah dijabarkan, maka dapat dirumuskan beberapa permasalahan sebagai berikut:

1. Bagaimana tingkat penggunaan CPU dan Memori dari penerapkan *load balancing* dengan menggunakan kontroler POX pada arsitektur *Software Defined Network.*
2. Bagaimana nilai *latency* dari penerapan *load balancing* dengan menggunakan kontroler POX pada arsitektur *Software Defined Network*.
3. Bagaimana nilai *throughput* dari penerapan *load balancing* dengan menggunakan kontroler POX pada arsitektur *Software Defined Network.*

1.3 Tujuan

Tujuan :

1. Untuk mengetahui tingkat penggunaan CPU dan Memori dari penerapkan *load balancing* dengan menggunakan kontroler POX pada arsitektur *Software Defined Network.*

2. Untuk mengatahui nilai *latency* dari penerapan *load balancing* dengan menggunakan kontroler POX pada arsitektur *Software Defined Network.*
3. Untuk mengatahui nilai *throughput* dari penerapan *load balancing* dengan menggunakan kontroler POX pada arsitektur *Software Defined Network*.

1.4 Manfaat

Manfaat dari penelitian ini adalah :

- Dapat menjadi refrensi implementasi *load balancing* kontroler.
- Dapat menjadi refrensi bahan penelitian selanjutnya yang berkaitan dengan kemampuan kinerja *load balancing* kontroler pada arsitektur *Software Defined Network* dalam sebuah jaringan.

1.5 Batasan masalah

Batasan masalah pada pengujian performa *load balancing* kontroler pada arsitektur *Software Defined Network* adalah sebagai berikut:

- Mengimplementasikan kontroler POX untuk mengontrol sebuah jaringan, tidak membandingkan dengan kontroler lain.
- Simulator *Cbench* digunakan sebagai tools standard untuk pengujian kontroler.

1.6 Sistematika pembahasan

Sistematika pembahasan dari penyusunan penelitian yang direncanakan adalah sebagai berikut:

BAB I PENDAHULUAN

Pendahuluan terdiri dari latar belakang, identifikasi dan pembatasan masalah, rumusan masalah, tujuan dan manfaat serta sistematika pembahasan dari penelitian ini. Selain itu juga sistematika pembahasan digunakan sebagai acuan untuk melakukan pembahasan terhadap penelitian yang dilakukan secara sistematis.

BAB II LANDASAN KEPUSTAKAAN

Bab landasan kepustakaan menjelaskan tentang teori – teori yang dipakai dalam penelitian, temuan dan bahan penelitian yang mungkin sebelumnya diperoleh dari beberapa refrensi yang menunjang penelitian dalam penulisan skripsi.

BAB III METODOLOGI PENELITIAN

Bab Metodologi penelitian menjelaskan tahapan - tahapan dan langkah – langkah dalam melakukan penelitian. Langkah – langkah seperti perancangan, implementasi, pengujian, serta analisis hasil dibahas secara umum. Selain itu pada bab metodologi juga dijelaskan secara singkat mengenai perangkat keras yang

dibutuhkan, arsitektur system, pengolahan, format data serta proses transmisi dalam penelitian.

BAB IV PERANCANGAN

Bab perancangan menjelaskan perancangan simulasi dan metode pengujian yang digunakan pada penerapan *load balancing* kontroler pada arsitektur *Software Defined Network.*

BAB V IMPLEMENTASI

Bab Implementasi menjelaskan tentang bagaimana menerapkan sistem yang telah dibuat berdasarkan sistematika sebelumnya secara rinci beserta dengan langkah-langkah pengerjaan yang dilakukan serta menampilkan gambar-gambar implementasi yang dilakukan.

BAB VI PENGUJIAN DAN ANALISIS HASIL PENGUJIAN

Bab Pengujian dan Analaisis Hasil Pengujian menjelaskan tentang pengujian terhadap sistem yang telah dibuat berdasarkan skenario yang telah dibuat pada bab sebelumnya. Serta menyajikan data hasil pengujian beserta analisis dari implementasi sistem yang telah dibangun. Hasil pengujian didapatkan dari pengujian yang telah dilakukan sesuai dengan skenario pengujian yang telah ditentukan. Dari hasil tersebut dapat dilakukan analisis terhadap kinerja sistem, serta mengetahui bagaimana sistem dapat mencapai tujuan.

BAB VII PENUTUP

Pada Bab penutup dipaparkan kesimpulan dari pelaksanaan penelitian. Kesimpulan ini dibuat dengan hasil pengujian dan analisis terhadapa perancangan dan implementasi arsitektur system sebagai dasarnya. Saran-saran juga diberikan agar hasil dari penelitian ini dapat diperbaiki dan disempurnakan apabila penelitian ini dikembangkan kemudian.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini berisi tinjauan pustaka yang meliputi kajian pustaka dan dasar teori yang diperlukan untuk penelitian. Kajian pustaka membahas penelitian yang telah ada dan yang diusulkan. Dasar teori membahas teori yang diperlukan untuk menyusun penelitian yang diusulkan, dasar teori meliputi teori tentang *Software Defined Network*, *OpenFlow*, *Load Balancing*, kontroler, *POX*, HAproxy, simulator *Cbench* beserta bagian pendukung pengujian yaitu PSUTIL, *flow setup rate* dan *flow setup delay*.

2.1 Kajian Pustaka

Kajian pustaka membahas penelitian yang telah ada dan yang diusulkan. Pada penelitian ini kajian pustaka diambil dari beberapa penelitian yang relevan yang pernah dilakukan. Dasar atau acuan yang berupa teori-teori atau temuan-temuan melalui hasil berbagai penelitian sebelumnya merupakan hal yang sangat perlu dan dapat dijadikan sebagai data pendukung. Salah satu data pendukung yang menurut peneliti perlu untuk dijadikan bagian tersendiri adalah penelitian terdahulu yang relevan dengan permasalah yang sedang dibahas dalam penelitian ini.

Penelitian pertama adalah penelitian yang dilakukan oleh Murdha dan Sawang (2015), fokus dari penelitian tersebut adalah bagaimana perbandingan performa yang dihasilkan dari penggunaan *single* server kontroler oleh dua kontroler berbeda yaitu POX dan Floodlight dimana pada dasarnya menggunakan bahasa pemrograman yang berbeda. POX dibangun dengan bahasa pemrograman Python dan berdasarkan arsitektur kontroler sebelumnya yaitu NOX, sedangkan Floodlight dibangun dengan bahasa pemrograman Java dan berdasarkan arsitektur kontroler Beacon.

Penelitian kedua adalah penelitian yang dilakukan oleh Mahdiyanah dan Naela (2016), fokus dari penelitian tersebut adalah bagaimana *load balancing* dengan menggunakan algoritma yang bersifat statis masih dapat berjalan dengan baik. Tujuan dari penelitian ini adalah agar *web server* dapat membagi bebannya secara optimal pada *Software Defined Network*. Hasil dari penelitian ini adalah *load balancing* dapat berjalan dengan baik dan membagi beban secara bergantian.

Penelitian ketiga adalah penelitian yang dilakukan oleh Jehn Ruey (2015) fokus dari penelitian tersebut adalah bagaimana penggunaan *load balancing* dengan *proposed algorithm* dibandingkan dengan *round-robin*, *randomized load balancing* dan LABERIO dibawah *Abilene network* dilihat dari *latency* dan *throughput*. Hasil yang didapatkan yaitu penggunaan *proposed algorithm* lebih unggul dibanding algoritma lainnya. Hal ini dapat dilihat dari path atau alur yang dipilih kontroler dan nilai *throughput* yang stabil dibandingkan algoritma lain. **Tabel 2.1** menjelaskan tentang kajian pustaka yang digunakan dalam penelitian.

Tabel 2.1 Kajian Pustaka

No	Judul	Penulis	Perbandingan	
			Kajian Pustaka	Tugas Akhir Penulis
1	Pengujian Performa Kontroler Software Defined Network (SDN): POX dan Floodlight	Sawang Murdha Anggara	Melakukan analisia perbandingan performa pada kontroler POX dan Floodlight.	Melakukan analisis terhadap *load balancing* kontroler POX *Software Defined Network.*
2	Implementasi Load Balancing di Web Server dengan Algoritma Round Robin pada Software Defined Network	Mahdiyanah, Wilda Naela	Mengimplementasikan sistem *load balancing* dengan menggunakan algoritma *Round Robin* pada *web server.*	Melakukan analisis terhadap *load balancing* kontroler POX pada *Software Defined Network.*
3	The Extend Dijkstra's-based Load Balancing for OpenFlow	Jiang Jehn-Ruey	Melakukan analisis terhadap *load balancing* dengan pengembangan algoritma Dijkstra untuk menemukan server terdekat.	Melakukan analisis terhadap *load balancing* kontroler POX pada *Software Defined Network.*

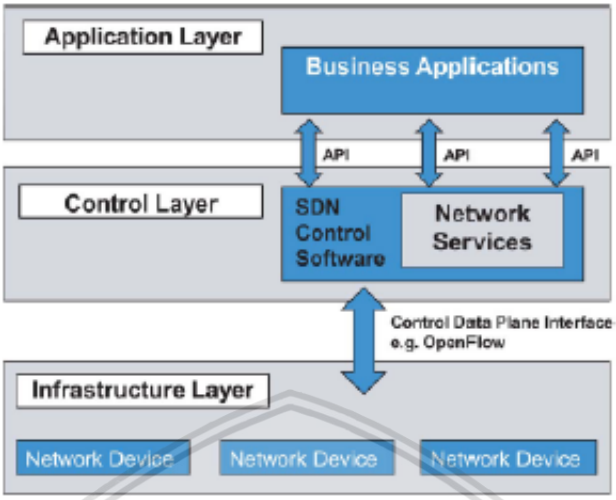
2.2 Software Defined Network

Software Defined Network (SDN) merupakan sebuah konsep atau paradigma dalam mendesain, mengelola dan mengimplementasikan jaringan yang semakin lama kebutuhan dan inovasi akan semakin kompleks. Konsep dari SDN sendiri adalah dengan melakukan pemisahan antara *control plane* dan data *plane* (Bailey, S., Bansal, D., dan Dunbar, L., 2013).

Beberapa hal penting dari *Software Defined Network* adalah :

- Ada pemisahan secara fisik antara *control plane* dan *data plane*.
- Memiliki antarmuka yang standard untuk melakukan pemrograman perangkat jaringan.

Arsitektur dari Software Defined Network sendiri dibagi menjadi 3 lapis bagian sebagaimana ditunjukkan pada **Gambar 2.1** dibawah ini.



Gambar 2.1 Arsitektur *Software Defined Network*

[Sumber : Hu, F., Qi Hao., dan Ke Bao., 2014]

Pada gambar diatas dapat dilihat bahwa arsitektur dari *Software Defined Network* terdapat beberapa bagian diantaranya:

- *Application layer*

  Berada pada lapis teratas dan bertujuan untuk berkomunikasi dengan sistem.

- *Control layer (control plane)*

  Mentraslasikan kebutuhan dari layer aplikasi menuju layer insfrastruktur dengan memberikan instruksi yang sesuai untuk SDN Datapath serta memberikan informasi yang relevan dan dibutuhkan oleh layer aplikasi.

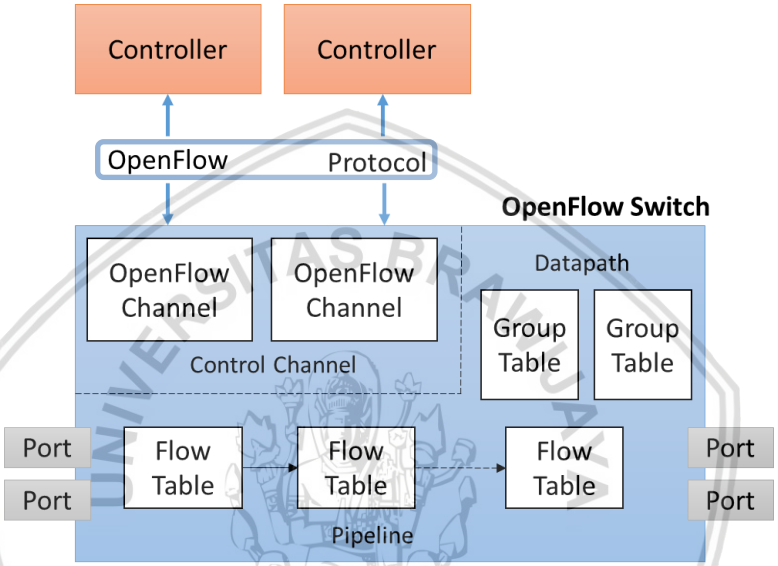
- *Infrastruktur layer (data plane)*

  Elemen jaringan yang mengatur SDN Datapath sesuai dengan instruksi yang diberikan melalui *Control-Data-Plane-Interface* (CDPI).

Setiap perangkat yang sudah mendukung *openflow* maka secara otomatis ketika diaktifkan akan memisahkan *control plane* dan *data plane,* lalu pada kontroler akan muncul virtual *network* atau virtual topologi yang menunjukkan setiap perangkat yang terhubung. Pemisahan *control plane* dan *data plane* diharapkan mampu mengatasi kompleksitas yang terjadi pada jaringan tradisional. *Software Defined Network* juga memungkinkan seorang administrator jaringan untuk mengendalikan *data flow* sesuai dengan perilaku jaringan.

2.3 OpenFlow

OpenFlow adalah protokol standard yang disediakan *Software Defined Network*. *OpenFlow* digunakan untuk melakukan komunikasi antara *control layer* dengan *infrastruktur layer* (Hu, F., Qi Hao., dan Ke Bao., 2014). Pada SDN terdapat pengendali berupa *software* yang dinamakan dengan kontroler dimana tugas dari kontroler ini adalah mengendalikan lalu lintas data pada infrastruktur jaringan seperti *switch* dan *router*. Kontroler ini berkomunikasi dengan infrastruktur jaringan menggunakan protokol *OpenFlow* seperti yang ditunjukkan pada **Gambar 2.2** berikut.



Gambar 2.2 *OpenFlow*

[sumber : grotto-networking.com]

Arsitektur jaringan OpenFlow terdiri dari beberapa komponen utama yaitu :

1. *OpenFlow Switch*

   Merupakan sebuah *forwarding device* yang memiliki beberapa komponen didalamnya yaitu :

   - *OpenFlow Channel* : menyediakan jalur untuk mentransmisikan perintah dan paket antara kontroler dan *switch.*
   - *Flow Table* : berisi *flow-entries* yang merubah paket masuk menjadi *flow* kemudian mengambil tindakan untuk paket tersebut.
   - *Group Table* : merupakan kumpulan tindakan yang dapat dilakuakan dan disesuaikan sesuai kebutuhan paket yang masuk.

2. *Flow-Entries*

   Menentukan bagaimana paket-paket diubah menjadi satu alliran (flow) yang selanjutnya akan dilakukan pemrosesan dan kemudian dilakukan *forwarding*. *Flow-Entries* memiliki komponen yaitu:

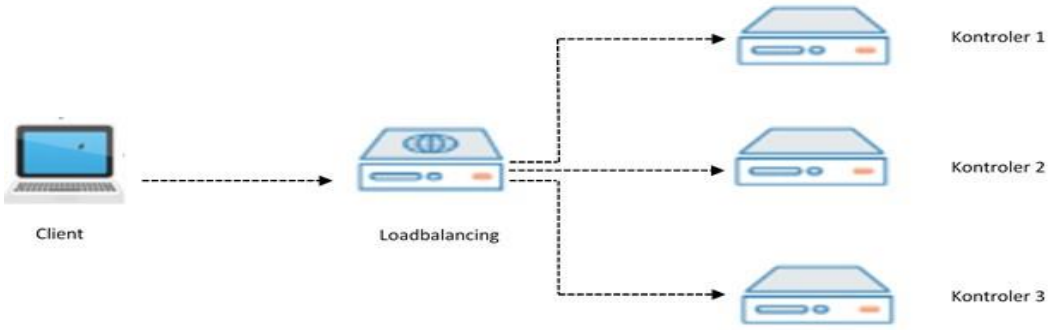
   - *Match Fields* : berfungsi untuk mencocokkan paket yang masuk berdasarkan informasi pada *header* paket, ingress port (jalan masuk), dan metadata.
   - *Counters* : berfungsi mengumpulkan data yang digunakan pada *flow* seperti jumlah paket diterima, kapasitas dan durasi dari satu *flow.*
   - *Instruction* : berisi tindakan atau instruksi yang akan digunakan pada match dan mengatur bagaimana cara mencocokkan paket-paket tersebut.

3. Kontroler

   Sebuah kontroler dapat melakukan pembaruan *flow-entries* pada *flow-table.*

2.4 Load Balancing

Load balancing merupakan tehnik pembagian beban yang digunakan dalam jaringan. Tehnik ini dilakukan dengan membagikan beban *traffic* pada paling sedikit dua jalur koneksi dengan pembagian beban seimbang. Tujuan dari pembagian beban secara merata ini untuk membuat traffic berjalan optimal, memaksimalkan *throughput,* mempercepat waktu respon dan menghindari adanya *overload* pada salah satu jalur koneksi. *Load balancing* digunakan ketika sebuah *server* memiliki jumlah user yang banyak atau dapat dikatakan melebihi batas maksimal kapasitasnya. *Load balancing* juga membagi beban kerja secara merata di dua atau lebih computer, link jaringan, *CPU* atau sumber daya lainnya untuk mendapatkan pemanfaatan *resource* yang optimal (Guo, Z., Su, M., dan Duan, Z., 2014). Diagram sederhana dari implementasi loadbalancing dapat dilihat pada **Gambar 2.3** berikut.



Gambar 2.3 *Loadbalancing* implementasi

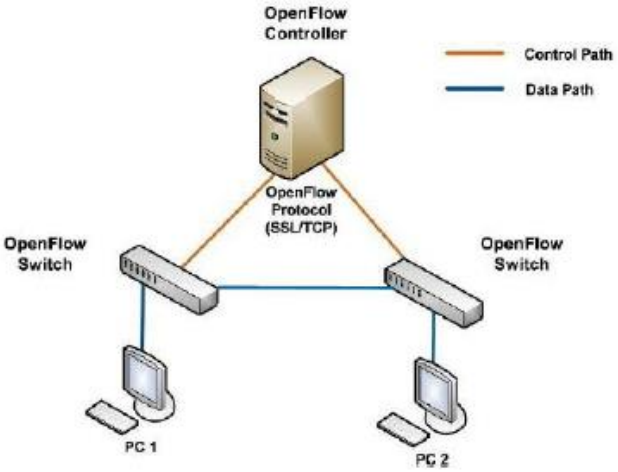
Pada **Gambar 2.3** diatas merupakan contoh dari *load balancing* yang diimplementasikan untuk kontroler dengan tujuan membagi beban agar merata dari masing-masing kontroler. *Load balancing* dapat diterapkan menggunakan beberapa macam jenis algoritma yang dipilih berdasarkan dari kondisi lingkungan jaringan, algoritma tersebut diantaranya:

1. *Round Robin*, merupakan algoritma dengan alur pembagian kerja secara berurutan sehingga setiap server akan mendapatkan giliran sesuai dengan urutannya.
2. *Ratio*, merupakan algoritma dengan pembagian beban sesuai dengan rasio masing-masing server. Server dengan rasio tinggi akan mendapatkan beban yang besar begitu juga sebaliknya, server dengan rasio kecil akan mendapatkan beban yang kecil.
3. *Fastest*, merupakan algoritma dengan mengutamakan server yang memiliki respon paling cepat.
4. *Least Connection*, merupakan algoritma pembagian kerja dengan melihat seberapa besar beban yang ditangani oleh server saat ini, jika server masih menangani jumlah yang sedikit maka untuk beban selanjutnya akan diberikan pada server tersebut.

Dengan memilih algoritma yang cocok dengan kondisi jaringan, maka kinerja dari *load balancing* sendiri juga semakin efisien karena setiap server yang diberikan tugas akan bekerja maksimal sesuai dengan tingkat kemampuan masing-masing. Pada penelitian ini algoritma yang digunakan adalah *Round Robin,* algoritma tersebut cocok diterapkan karena kontroler yang digunakan semuanya adalah POX sehingga akan memiliki performa yang sama.

2.5 Kontroler

Kontroler merupakan node yang bertindak sebagai pusat logika dari suatu jaringan. Semua pengaturan dan pengendalian jaringan seperti *routing*, *switching*, dan QoS berada dalam sebuah kontroler. Sehingga dapat dikatakan kontroler sebagai pusat kecerdasan dari sebuah jaringan. Dalam implementasinya kontroler merupakan sebuah unit perangkat lunak yang ditanam pada komputer *server* dan terpisah dari *switch*. Seperti pada **Gambar 2.4** dibawah ditunjukkan sebuah kontroler berada terpisah dari *switch*.



Gambar 2.4 Kontroler dalam *Software Defined Network*

[Sumber : Sonba, A., dan Hassan, A., 2014]

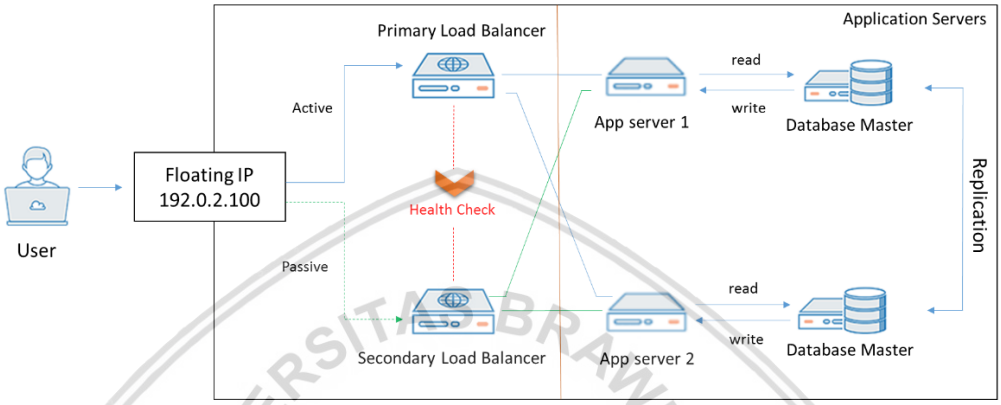
2.6 HAproxy

High Availability Proxy (HAproxy) merupakan *software opensource* yang popular dimana menawarkan solusi yang handal untuk mengatasi *High Availability Proxy* dan *loadbalancing* TCH/HTTP. HAproxy ini paling umum digunakan untuk meningkatkan kinerja dan kehandalan dari lingkungan server dengan cara mendistribusikan beban kerja ke beberapa server (seperti web, aplikasi, database). Beberapa sistem operasi / platforms yang dapat menjalankannya adalah sebagai berikut:

- Linux 2.4 on x86, x86_64, Alpha, Sparc, MIPS, PARISC
- Linux 2.6 / 3.x on x86, x86_64, ARM, Sparc, PPC64
- Solaris 8/9 on UltraSPARC 2 dan 3
- Solaris 10 on Opteron dan UltraSPARC
- FreeBSD 4.10 – 10 on x86
- OpenBSD 3.1 to – current on i386, amd 64, macppc, alpha, sparc dan VAX
- AIX 5.1 – 5.3 on Power Architecture

HAproxy menggunakan *Health Check* untuk menentukan mana *backend* server yang tersedia untuk menerima *request. Health Check* ini dilakukan dengan cara membuat koneksi TCP ke server, apakah *backend* server sedang listening pada alamat IP dan Port yang telah dikonfigurasikan atau malah sebaliknya. Hal ini dapat mengantisipasi jika salah satu server yang secara tiba tiba mati. Jika *Health Check* mendeteksi ada *backend* server yang mati dan tidak bisa melayani *request* maka secara otomatis server tersebut akan dinonaktifkan dan lalu lintas data tidak akan diteruskan ke server tersebut hingga sudah kembali ke kondisi baik.

Jika menggunakan konsep simple *load balancing* seperti maka masih memungkinkan adanya *single point of failure* ketika server *loadbalancing* yang mengalami kerusakan dan harus mati sehingga mengakibatkan *high latency* dan *downtime*, oleh karena itu HAproxy menawarkan solusi High Availability untuk menghilangkan *single point of failure* tersebut. *Loadbalancing* memang menjadi solusi untuk menyeimbangkan beban *backend* server menjadi sama rata, namun persiapan High Availability yang benar adalah dengan menyiapkan server *loadbalancing* yang dibangun lebih dari satu seperti **Gambar 2.5** dibawah ini.



Gambar 2.5 *High Availability Loadbalancing* model

[Sumber : www.digitalocean.com]

Pada contoh diatas dibuat *multiple loadbalancing,* satu diataranya aktif dan lainnya pasif. Ketika akan mengakses suatu app server maka *request* akan melalui server *loadbalancing* yang aktif terlebih dahulu, dan ketika server *loadbalancing* tersebut gagal menangani *request,* maka akan terdeteksi oleh mekanisme *failover* dan secara otomatis memberikan request tersebut ke server *loadbalancing* yang pasif sehingga tidak akan terjadi *single point of failure.*

2.7 POX

POX adalah *platform open-source* yang digunakan untuk *Software Defined Network.* POX sendiri menggunakan bahasa pemrograman Python yang dibangun berdasarkan kontroler pendahulunya yaitu NOX. Kontroler POX sendiri juga memungkinkan proses perancangan dan pembangunan jaringan menjadi lebih cepat dan lebih umum dibandingkan dengan NOX (Al-Shabibi, A., dan McCauley, M., 2015).

POX membutuhkan versi minimal Python yaitu 2.6. POX sendiri juga dapat dijalankan pada sistem operasi Windows, Mac OS, dan Linux. POX dapat digunakan dengan "standar" Pythin interpreter (CPython), tetapi juga mendukung PyPy.

Fitur yang terdapat dalam kontroler POX sendiri adalah sebagai berikut:

a. Dapat dijalankan di semua sistem.

b. Mendukung tampilan GUI dan visualisasi yang sama seperti NOX.

c. Memiliki kinerja yang lebih baik dibanding NOX.

d. Dioperasikan untuk sistem operasi Linux, Mac OS, dan Windows.

e. Contoh komponen yang dapat digunakan kembali untuk seleksi jalur, penemuan topologi, dan lainnya.

2.8 Cbench

Cbench adalah *benchmarking tool* standar yang digunakan untuk melakukan pengukuran terhadap *throughput* dan *latency* pada sebuah kontroler (Muntaner, Guillermo R.d.T. 2012). *Cbench* juga bagian dari *Oflops* (*OpenFlow per second*) yaitu sebuah tool yang berfungsi untuk mengukur aspek khusus pada *switch OpenFlow*. Pengembangannya dilakukan dengan menggunakan Bahasa pemrograman C. Pada *Cbench* ini terdapat 2 mode yaitu:

- *Throughput mode*: *switch* yang diemulasikan mengirimkan sebanyak mungkin paket data kepada kontroler. *Cbench* kemudian akan menghitung respon yang diberikan oleh kontroler sebagai balasan paket data yang telah dikirimkan oleh *switch*.
- *Latency mode*: *swith* diemulasikan untuk mengirimkan paket tunggal kepada kontroler, kemudian *Cbench* menghitung waktu saat paket dikirim ke kontroler hingga paket kembali diterima oleh *switch*.

2.9 Metode Pegujian

Pada sub bab ini akan dijelaskan metode pengujian yang akan digunakan dalam penelitian ini, terdapat 3 metode pengujian untuk menguji tingkat penggunaan CPU dan Memori, nilai latency dan throughput yaitu:

- PSUTIL
- Flow Setup Rate
- Flow Setup Delay

2.9.1 PSUTIL

Proses and system utilities (psutil) adalah sebuah *closs-platform library* didalam Python untuk mengambil informasi dari sebuah proses yang sedang berjalan dan penggunaan sistem seperti CPU, *memory, disk, network, sensor*. *Library* ini sangat berguna untuk memantau sistem dan pengelolaan proses yang sedang berjalan. Pada psutil mendukung beberapa platform seperti berikut:

- Linux
- Windows
- OSX
- FreeBSD, Open BSD, NetBSD
- Sun Solaris

- AIX

Dalam penelitian ini akan menggunakan psrecord. Dimana psrecord ini berjalan dengan menggunakan library dari psutil untuk merekam log dari penggunakan CPU dan memori pada saat *loadbalancing* dijalankan.

2.9.2 Flow Setup Rate

Flow setup rate adalah jumlah maksimal respon yang mampu diberikan oleh kontroler dan ditunjukkan dalam satuan detik. Pengukuran dapat dilakukan dalam dua mode yaitu:

- *Constant network load*
- *Incremental network load*

Karakteristik *flow setup rate* dengan mode *constant network load* adalah jumlah iterasi pengujian yang bervariasi dan dilakukan menggunakan jumlah *switch* yang tetap. Sedangkan karakteristik *flow setup rate* dengan mode *incremental network load* merupakan kebalikannya, dimana jumlah iterasi pengujian yang tetap tetapi jumlah *switch* dapat ditambah pada setiap tahap pengujian (Vengainathan, B ., Anton, B., dan Vishwas, M., 2014).

2.9.3 Flow Setup Delay

Flow setup delay adalah waktu yang dibutuhkan oleh sebuah kontroler dalam memberikan respon *flow* yang ditunjukkan dalam satuan mili detik. Pengukuran dapat dilakukan dalam tiga mode yaitu:

- *Constant network load*
- *Incremental netwok load*
- *Stress network load*

Karakteristik *flow setup delay* dengan mode *constant network load* adalah jumalah iterasi pengujian yang bervariasi dan dilakukan menggunakan jumlah *switch* yang tetap. Karakteristik kedua adalah *incremental network load* dimana jumlah iterasi pengujian tetap tetapi jumlah dari *switch* dapat ditambah pada setiap pengujiannya. Karakteristik ketiga merupakan penggabungan dari keduanya yaitu *stress network load* dimana setiap pengujian diberikan iterasi yang semakin bertambah dan jumlah *switch* dapat bervariasi (Vengainathan, B ., Anton, B., dan Vishwas, M., 2014).

BAB 3 METODOLOGI

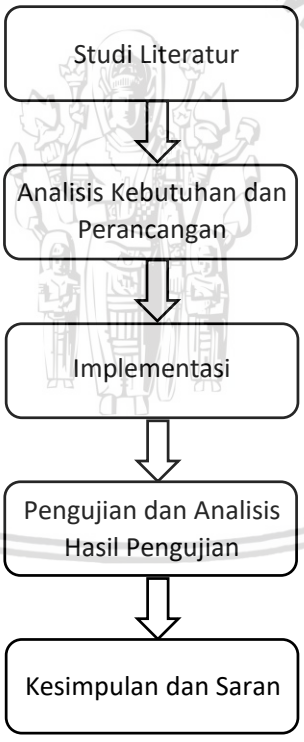
Bab ini membahas metodologi yang digunakan pada penelitian analisis performa *load balancing* kontroler menggunakan kontroler POX sebagai komponen utama pada *software defined network*.

3.1 Jenis Penelitian

Jenis penelitian yang dilakukan oleh penulis adalah penelitian implementatif dan pengembangan. Penelitian jenis ini dalam rangka mengimplementasikan dan menganalisis performa penerapan *load balancing* kontroler pada arsitektur *Software Defined Network*, penelitian dimulai dengan analisis kebutuhan, perancangan sistem, serta pengujian dari sistem yang telah dibuat.

3.2 Metodologi Penelitian

Metodologi penelitian yang dilakukan pada penelitian ini secara umum ditunjukkan pada **Gambar 3.1** sebaga berikut.



Gambar 3.1 Metode Penelitian

3.2.1 Studi Literatur

Studi literatur dilakukan bertujuan untuk memperlajari serta memahami konsep-konsep sistem agar ketika dilakukan perancangan tidak terlalu mengalami kendala. Jenis literature yang digunakan adalah artikel jurnal. Adapun yang perlu menjadi bahan studi literature pada penelitian ini meliputi :

A. *Software Defined Network*
B. *OpenFlow*
C. *Load Balancing*
D. *Kontroler*
E. *HAproxy*
F. *POX*
G. *Cbench*
H. *Flow Setup Rate*
I. *Flow Setup Delay*

3.2.2 Analisis Kebutuhan

Pada tahap ini merupakan tahap menganalisis kebutuhan sistem yang akan digunakan. Analisis kebutuhan sistem diperlukan agar mengetahui hal-hal yang diperlukan dalam pengembangan sistem untuk menghindari penggunaan sumberdaya yang tidak perlu, analisis kebutuhan juga digunakan untuk acuan dalam merancang sistem, sehingga perancangan nantinya dapat terbentuk secara sistematis dan terarah. Dalam melakukan analisis kebutuhan salah satu cara yang dapat digunakan adalah melalui kepustakaan yang telah ada, kepustakaan berlandaskan pada penelitian-penelitian sejenis yang telah dilakukan untuk mengetahui perangkat apa saja yang dapat digunakan dalam pengembangan sistem yang akan dibuat selanjutnya.

3.2.3 Perancangan

Perancangan dilakukan setelah semua kebutuhan didapatkan melalui analisis kebutuhan, perancangan dilakukan agar penelitian yang akan dilakukan dapat berjalan dengan baik dan sistematis. Tujuannya untuk memberikan gambaran bagaimana implementasi perangkat lunak dan perancangan jaringan.

Pada tahap perancangan perangkat lunak, dirancang sistem yang terdiri dari konfigurasi *virtual machine* dan kontroler yang nantinya ada 4 virtual machine yang dibuat. Dimana ada 3 kontroler yang masing-masing akan diinstall pada virtual machine yang berbeda, dan 1 simulator Cbench untuk memonitoring *throughput* dan *latency*. Selain itu juga ada perancangan konfigurasi *loadbalancing* menggunakan HAproxy.

Pada tahap perancangan jaringan, akan digambarkan bagaimana rancangan arsitektur jaringan yang menjelaskan konfigurasi IP dan komunikasi yang nantinya

akan digunakan untuk melakukan pertukaran data, selain itu juga ada rancangan koneksi kontroler dan HAproxy dimana ketiga kontroler yang berada pada *virtual machine* harus dapat terkoneksi dengan HAproxy untuk menjalankan *High Availability* dan *Loadbalancing.*

3.2.4 Implementasi

Implementasi dilakukan berdasarkan perancangan yang dibuat sebelumnya, maka dari itu implementasi akan mengacu pada tahapan – tahapan sebagai berikut:

1. Implementasi perangkat lunak, pengimplementasian perangkat lunak meliputi beberapa implementasi sebagai berikut:
    - Implementasi konfigurasi *virtual machine. Virtual Machine (*VM*)* digunakan sebagai mesin tambahan yang akan diintall dengan kontroler *Software Defined Network.* VM yang dibutuhkan sebanyak 4 buah menyesuaikan dengan jumlah kontroler dan simulator Cbench.
    - Implementasi konfigurasi kontroler. Dalam tahap ini kontroler *Software Defined Network* yang digunakan adalah POX, dimana akan dikonfigurasikan pada ketiga VM yang telah siap.
    - Implementasi konfigurasi *load balancing* menggunakan HAproxy. Tahap ini HAproxy akan dikonfigurasikan dan disiapkan untuk melakukan *load balancing* terhadap kontroler yang nantinya akan diakses oleh client.
    - Implementasi monitoring *throughput* dan *latency* dari Cbench. Tahap ini simulator Cbench akan dikonfigurasikan pada VM client, sehingga nantinya dapat dilakukan pengujian terhadap *loadbalancing* kontroler.
2. Implementasi jaringan, pengimplementasian jaringan meliputi beberapa implementasi sebagai berikut:
    - Implementasi arsitektur jaringan. Pada tahap ini setiap perangkat akan diberikan alamat IP dan dikonfigurasikan agar dapat terhubung antar satu sama lainnya sehingga dapat melakukan pertukaran data sesuai dengan topologi yang telah dibuat.
    - Implementasi koneksi kontroler dengan HAproxy. Dalam tahap ini kontroler yang telah dikonfigurasi didalam VM akan dikoneksikan dengan HAproxy, yang nantinya oleh HAproxy akan dijalankan fitur *High Availability* dan *Loadbalancing* ketika kontroler mendapatkan request dari client.

3.2.5 Pengujian dan Analisis Hasil Pengujian

Pada tahap ini dilakukan pengujian untuk mengetahui performa yang dihasilkan dari *load balancing* kontroler jika diterapkan pada sebuah jaringan dengan parameter yang telah ditentukan yaitu *flow setup delay*, *flow setup rate,* CPU dan Memori *Usage*. Pengujian akan dilakukan sesuai skenario yang telah ditetapkan serta nantinya jika sudah diketahui hasilnya maka akan dilakukan analisis dari hasil pengujian multikontroler pada *software defined network.* Pengujian yang dilakukan sebagai berikut:

- Flow Setup Delay

  Pengujian *Flow Setup Delay* atau *latency mode* ini akan dijalankan menggunakan simulator Cbench, dimana secara *default* simulator Cbench sudah berjalan pada *mode latency.* Tujuan dari pengujian ini adalah untuk mengetahui tingkat kecepatan respon per detik untuk setiap switch yang dikontrol yang dihasilkan dari masing-masing kontroler dengan jumlah kontroler antara 1 hingga 3 menggunakan *loadbalancing* dari HAproxy. Variable yang perlu dimasukkan untuk melakukan pengujian latency pada simulator Cbench sesuai dengan skenario pengujian berikut.

Tabel 3.1 Pengujian Flow Setup Delay 1 Kontroler dengan jumlah switch bervariasi dan host tetap

Uji Latency 1 Kontroler POX	
Parameter	**Nilai**
IP (-c)	192.168.56.101
Port (-p)	6633
Waktu (-m)	1000
Iterasi (-l)	5
Host (-M)	5
Variasi Switch (-s)	100,200,300,400,500,600,700,800,900,1000.
Mode (-t)	Tidak Ditulis

Tabel 3.2 Pengujian Flow Setup Delay 1 Kontroler dengan jumlah host bervariasi dan switch tetap

Uji Latency 1 Kontroler POX	
Parameter	**Nilai**
IP (-c)	192.168.56.101
Port (-p)	6633
Waktu (-m)	1000

Iterasi (-l)	5
Variasi Host (-M)	2000,4000,6000,8000,10000.
Switch (-s)	15
Mode (-t)	Tidak Ditulis

Tabel 3.3 Pengujian Flow Setup Delay 2 Kontroler dengan jumlah switch bervariasi dan host tetap

Uji Latency 2 Kontroler POX	
Parameter	**Nilai**
IP (-c)	192.168.56.1
Port (-p)	6633
Waktu (-m)	1000
Iterasi (-l)	5
Host (-M)	5
Variasi Switch (-s)	100,200,300,400,500,600,700,800,900,1000.
Mode (-t)	Tidak Ditulis

Tabel 3.4 Pengujian Flow Setup Delay 2 Kontroler dengan jumlah host bervariasi dan switch tetap

Uji Latency 2 Kontroler POX	
Parameter	**Nilai**
IP (-c)	192.168.56.1
Port (-p)	6633
Waktu (-m)	1000
Iterasi (-l)	5
Variasi Host (-M)	2000,4000,6000,8000,10000.
Switch (-s)	15
Mode (-t)	Tidak Ditulis

Tabel 3.5 Pengujian Flow Setup Delay 3 Kontroler dengan jumlah switch bervariasi dan host tetap

Uji Latency 3 Kontroler POX	
Parameter	**Nilai**
IP (-c)	192.168.56.1
Port (-p)	6633
Waktu (-m)	1000
Iterasi (-l)	5
Host (-M)	5
Variasi Switch (-s)	100,200,300,400,500,600,700,800,900,1000.
Mode (-t)	Tidak Ditulis

Tabel 3.6 Pengujian Flow Setup Delay 3 Kontroler dengan jumlah host bervariasi dan switch tetap

Uji Latency 3 Kontroler POX	
Parameter	**Nilai**
IP (-c)	192.168.56.1
Port (-p)	6633
Waktu (-m)	1000
Iterasi (-l)	5
Variasi Host (-M)	2000,4000,6000,8000,10000.
Switch (-s)	15
Mode (-t)	Tidak Ditulis

Pada skenario diatas akan dilakukan pengujian dengan langkah-langkah sebagai berikut:

1. Untuk *single* kontroler pengujian dilakukan tanpa menggunakan HAproxy. Untuk dua dan tiga kontroler pengujian dilakukan menggunakan HAproxy.
2. Setelah poin 1 sudah dipersiapkan, selanjutnya jalankan masing-masing kontroler pada setiap mesin yang sudah disiapkan.
3. Setelah kontroler siap maka jalankan simulator Cbench untuk memulai pengujian.

- Flow Setup Rate

Pengujian *Flow Setup Rate* atau *throughput mode* ini akan dijalankan menggunakan simulator Cbench, dimana dengan menambah variabel -t pada simulator Cbench agar berjalan pada *mode throughput.* Tujuan dari pengujian ini adalah untuk mengetahui berapa banyak jumlah flow yang dapat ditangani berdasarkan besar arus data (flow) yang dapat dikelola oleh masing-masing kontroler setiap detiknya untuk setiap host yang dikelola. Variable yang perlu dimasukkan untuk melakukan pengujian throughput pada simulator Cbench sesuai dengan skenario pengujian berikut.

Tabel 3.7 Pengujian Flow Setup Rate 1 Kontroler dengan jumlah switch bervariasi dan host tetap

Uji Throughput 1 Kontroler POX	
Parameter	**Nilai**
IP (-c)	192.168.56.101
Port (-p)	6633
Waktu (-m)	1000
Iterasi (-l)	5
Host (-M)	5
Variasi Switch (-s)	100,200,300,400,500,600,700,800,900,1000.
Mode (-t)	Ditulis

Tabel 3.8 Pengujian Flow Setup Rate 1 Kontroler dengan jumlah switch tetap dan host bervariasi

Uji Throughput 1 Kontroler POX	
Parameter	**Nilai**
IP (-c)	192.168.56.101
Port (-p)	6633
Waktu (-m)	1000
Iterasi (-l)	5
Variasi Host (-M)	2000,4000,6000,8000,10000.
Switch (-s)	15
Mode (-t)	Ditulis

Tabel 3.9 Pengujian Flow Setup Rate 2 Kontroler dengan jumlah switch bervariasi dan host tetap

Uji Throughput 2 Kontroler POX	
Parameter	**Nilai**
IP (-c)	192.168.56.1
Port (-p)	6633
Waktu (-m)	1000
Iterasi (-l)	5
Host (-M)	5
Variasi Switch (-s)	100,200,300,400,500,600,700,800,900,1000.
Mode (-t)	Ditulis

Tabel 3.10 Pengujian Flow Setup Rate 2 Kontroler dengan jumlah switch tetap dan host bervariasi

Uji Throughput 2 Kontroler POX	
Parameter	**Nilai**
IP (-c)	192.168.56.1
Port (-p)	6633
Waktu (-m)	1000
Iterasi (-l)	5
Variasi Host (-M)	2000,4000,6000,8000,10000.
Switch (-s)	15
Mode (-t)	Ditulis

Tabel 3.11 Pengujian Flow Setup Rate 3 Kontroler dengan jumlah switch bervariasi dan host tetap

Uji Throughput 3 Kontroler POX	
Parameter	**Nilai**
IP (-c)	192.168.56.1
Port (-p)	6633
Waktu (-m)	1000

Iterasi (-l)	5
Host (-M)	5
Variasi Switch (-s)	100,200,300,400,500,600,700,800,900,1000.
Mode (-t)	Ditulis

Tabel 3.12 Pengujian Flow Setup Rate 3 Kontroler dengan jumlah switch tetap dan host bervariasi

Uji Throughput 3 Kontroler POX	
Parameter	**Nilai**
IP (-c)	192.168.56.1
Port (-p)	6633
Waktu (-m)	1000
Iterasi (-l)	5
Variasi Host (-M)	2000,4000,6000,8000,10000.
Switch (-s)	15
Mode (-t)	Ditulis

Pada skenario diatas akan dilakukan pengujian dengan langkah-langkah sebagai berikut:

1. Untuk *single* kontroler pengujian dilakukan tanpa menggunakan HAproxy. Untuk dua dan tiga kontroler pengujian dilakukan menggunakan HAproxy.
2. Setelah poin 1 sudah dipersiapkan, selanjutnya jalankan masing-masing kontroler pada setiap mesin yang sudah disiapkan.
3. Setelah kontroler siap maka jalankan simulator Cbench untuk memulai pengujian.

- CPU dan Memori *Usage*

Pengujian CPU dan Memori *usage* ini akan dijalankan menggunakan library psutil yang dijalankan oleh psrecord, dimana psrecord akan mengambil informasi penggunaan berupa log sesuai dengan variabel yang digunakan. Tujuan dari pengujian ini adalah untuk mengetahui seberapa besar penggunaan CPU dan Memori pada masing-masing server ketika lingkungan sistem dijalankan untuk melakukan *loadbalancing.* Variable yang perlu dimasukkan untuk melakukan pengujian CPU dan Memori *usage* oleh psrecord sesuai dengan skenario pengujian berikut.

Tabel 3.13 Pengujian CPU dan Memori *Usage* 1 Kontroler

CPU dan Memori *Usage* 1 Kontroler	
Parameter	**Nilai**
PID	PID POX
File	nama_file.txt
Durasi	15
Interval	1
Variasi Host (-M)	2000,4000,6000,8000,10000.
Switch (-s)	15
Mode (-t)	Tidak Ditulis

Tabel 3.14 Pengujian CPU dan Memori *Usage* 2 Kontroler dan LB server

CPU dan Memori *Usage* 2 Kontroler dan LB server	
Parameter	**Nilai**
PID	PID HAproxy dan PID POX
File	nama_file.txt
Durasi	15
Interval	1
Variasi Host (-M)	2000,4000,6000,8000,10000.
Switch (-s)	15
Mode (-t)	Tidak Ditulis

Tabel 3.15 Pengujian CPU dan Memori *Usage* 3 Kontroler dan LB server

CPU dan Memori *Usage* 3 kontroler dan LB server	
Parameter	**Nilai**
PID	PID HAproxy dan PID POX
File	nama_file.txt
Durasi	15
Interval	1
Variasi Host (-M)	2000,4000,6000,8000,10000.
Switch (-s)	15
Mode (-t)	Tidak Ditulis

Pada skenario diatas akan dilakukan pengujian dengan langkah-langkah sebagai berikut:

1. Untuk *single* kontroler pengujian dilakukan tanpa menggunakan HAproxy. Untuk dua dan tiga kontroler pengujian dilakukan menggunakan HAproxy.
2. Setelah poin 1 sudah dipersiapkan, selanjutnya jalankan masing-masing kontroler pada setiap mesin yang sudah disiapkan.
3. Setelah kontroler siap maka jalankan psutil.

3.2.6 Pengambilan Kesimpulan dan Saran

Pengambilan kesimpulan dan saran dilakukan setelah semua tahapan perancangan implementasi, dan pengujian sistem yang dibangun telah selesai dilakukan. Kesimpulan diambil dari hasil pengujian dan analisis terhadap sistem yang dibangun serta kelebihan dan kekurangan sistem yang telah dibuat, dan kesesuaian sistem antara teori dan praktik, yang nantinya akan menjawab rumusan masalah yang telah dirumuskan sebelumnya. Tahap terakhir dari penulisan adalah saran yang dimaksudkan untuk memberikan pertimbangan atas pengembangan penelitian lebih lanjut.

BAB 4 ANALISIS KEBUTUHAN DAN PERANCANGAN

Analisis kebutuhan dan perancangan diperlukan agar implementasi dapat berjalan dengan sistematis, dalam analisis kebutuhan berisi tentang perangkat – perangkat yang dibutuhkan dalam implementasi meliputi kebutuhan perangkat keras dan perangkat lunak. Sementara perancangan digunakan sebagai acuan yang digunakan untuk melakukan implementasi sehingga dapat sesuai dan terarah. Perancangan ini terdiri atas perancangan perangkat keras, perancangan jaringan dan perancangan perangkat lunak. Perancangan perangkat keras akan menggambarkan integrasi antara perangkat keras, perancangan jaringan akan menggambarkan arsitektur dan koneksi pada jaringan, sedangkan perancangan perangkat lunak akan menggambarkan *tools* yang digunakan untuk konfigurasi dan memonitoring sistem.

4.1 Analisis Kebutuhan

Untuk melakukan implementasi *loadbalancing* kontroler maka diperlukan *Virtual Machine* yang didalamnya akan dibangun 3 virtual kontroler sebagai server dan 1 virtual client, ketiga virtual kontroler ini nantinya akan terintegrasi dengan virtual *client* yang berisi simulator Cbench dimana akan memanipulasi jumlah *host* dan *switch* yang akan mengirimkan data. Data yang dikirimkan oleh *client* akan mengarah pada HAproxy dimana nantinya akan dilakukan *loadbalancing* yang diarahkan pada ketiga virtual kontroler agar ditangani secara bergantian sesuai dengan skenario yang telah dibuat. Oleh karena itu kebutuhan perangkat keras dan perangkat lunak dalam membangun lingkungan jaringan tersebut terdiri dari:

a. Perangkat keras

   Sebuah laptop dengan spesifikasi :

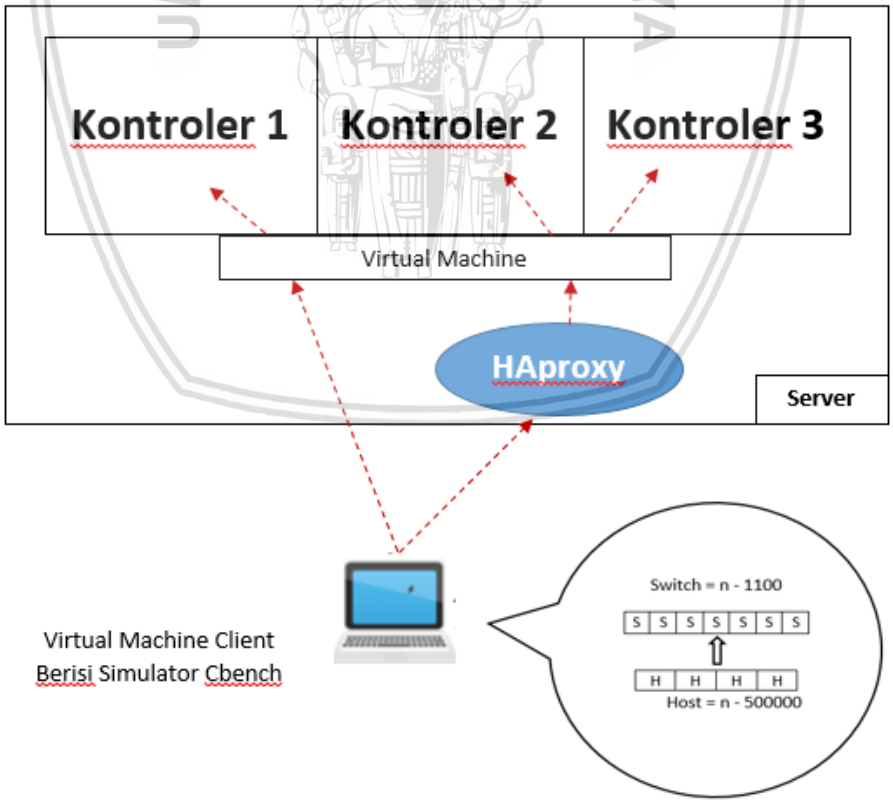
   - Pada Host yang akan dipasang HAproxy dan *Virtual Machine* sebagai berikut:
     a. Intel(R) Core(TM) i5-4200U – CPU@1,60 GHz
     b. RAM 4 GB
     c. Harddisk 500 GB
     d. OS Ubuntu 14.04
   - *Virtual Machine* yang akan digunakan dengan spesifikasi sebagai berikut:
     a. CPU Virtual 4 cores
     b. RAM Virtual 521 MB
     c. OS Ubuntu Server 14.04

b. Perangkat lunak

1. *Virtual Machine*
2. HAproxy
3. Kontroler POX
4. Simulator Cbench

4.2 Perancangan

Pada penelitian ini akan dibuat contoh sederhana mengenai *loadbalancing* kontroler POX dimana jumlah kontroler yang digunakan sebanyak 3, dan masing-masing kontroler ini akan dikonfigurasikan pada *Virtual Machine*. Kontroler ini akan berada pada satu laptop yang sama dengan HAproxy sebagai server, bedanya HAproxy tidak berada dalam *Virtual Machine* tetapi diluar. Ketiga kontroler yang sudah siap akan dikoneksikan dengan HAproxy untuk menjalankan fitur *High Availability* dan *Loadbalancing* dari HAproxy. Sedangkan Cbench akan dikonfigurasikan pada *Virtual Machine Client* dimana nantinya dapat memberikan simulasi request menuju kontroler dari beberapa *host* dan *switch* yang dapat diatur sesuai kebutuhan pengujian. Penggambaran arsitektur *loadbalancing* kontroler *Software Defined Network* yang dibuat pada penelitian ini dapat dilihat pada **Gambar 4.1** berikut:



Gambar 4.1 Arsitektur *Loadbalancing* kontroler *Software Defined Network*

Dilihat dari topologi diatas terdapat 3 kontroler yang telah diintall pada *Virtual Machine* akan terkoneksi dengan HAproxy, dibuat 3 virtual kontroler tujuannya agar kontroler tersebut membagi beban yang diterima dari *client* secara bergantian sehingga dapat melayani jumlah *request* yang banyak. Sedangkan tujuan dari terkoneksinya kedua kontroler dengan HAproxy adalah untuk mengimplementasikan *loadbalancing* yang merupakan fitur yang disediakan oleh HAproxy. Selain itu HAproxy juga menyediakan fitur *High Availability* dimana nantinya dapat memudahkan pembagian kerja jika ada salah satu kontroler yang mati.

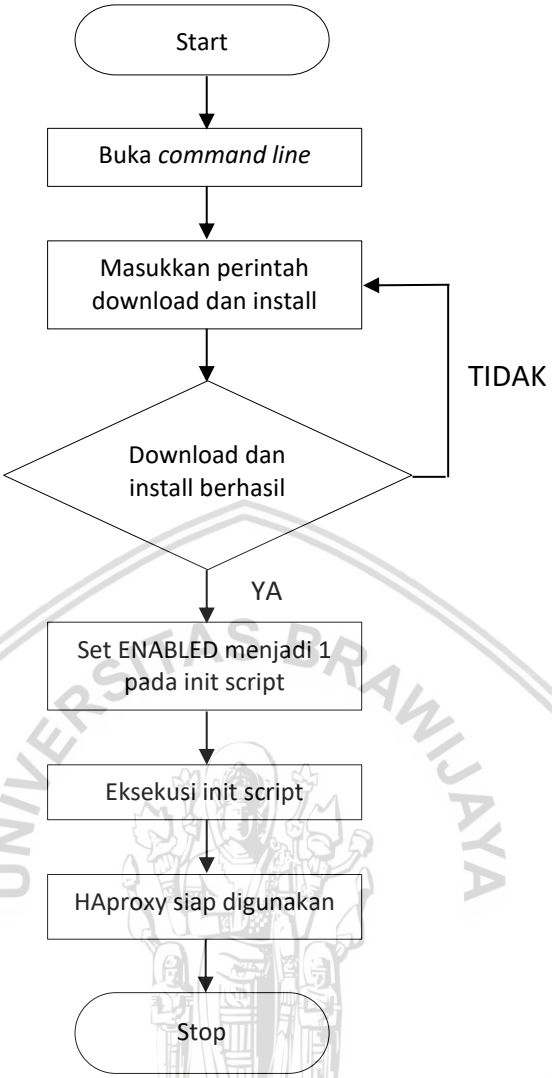
Pada *VM Client* sendiri akan dipasang simulator Cbench yang akan mengirimkan beberapa manipulasi request yang dikirimkan ke HAproxy dan diteruskan menuju kedua virtual kontroler. Pada simulator Cbench sendiri kita dapat memasukkan beberapa perintah seperti alamat IP yang dituju, port yang dituju, berapa banyak *host* dan *switch* yang akan digunakan selama proses pengujian dan masih banyak lainnya.

4.2.1 Perancangan Perangkat Lunak

Sebelum menghubungkan antar perangkat untuk menjadi sebuah sistem yang akan menjalankan fungsi *loadbalancing* kontroler *Software Defined Network*, terlebih dahulu kita siapkan beberapa perangkat lunak yang harus diinstall yaitu *virtual machine*, kontroler, HAproxy dan Cbench *simulator.* Terlebih dulu *virtual machine* harus diinstall untuk memberikan mesin tambahan yang didalamnya akan diintall kontroler POX, lalu HAproxy akan diintal di laptop host sehingga letak kontroler dan HAproxy tidak dalam satu mesin yang sama. Setelah keduanya dikonfigurasikan maka selanjutnya pada VM client kita pasang simulator Cbench. Konfigurasi yang berbeda-beda diperlukan dalam mempersiapkan perangkat lunak yang digunakan, maka dari itu dibutuhkan perancangan perangkat lunak.

4.2.1.1 Perancangan Konfigurasi Loadbalancing dari HAproxy

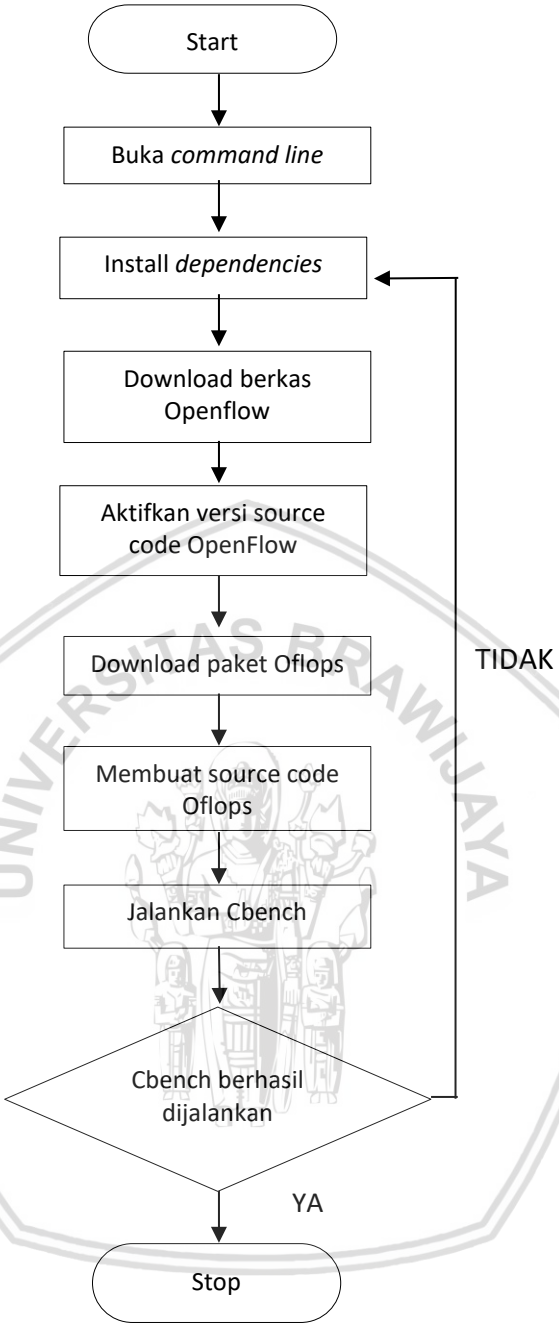
Perancangan monitoring loadbalancing dari HAproxy diperlukan untuk melakukan pengujian, dengan fitur yang dimiliki HAproxy yaitu *loadbalancing* dan *high availability* akan memaksimalkan kinerja dari kontroler ketika menerima *request* dari banyak *client.* Pada HAproxy sendiri juga dapat dilihat koneksi kontroler, berapa banyak request yang dilayani oleh setiap kontroler dan lain-lain. Alur perancangan monitoring loadbalancing dari HAproxy dapat dilihat pada **Gambar 4.2** berikut:



Gambar 4.2 Perancangan Monitoring *Loadbalancing* dari HAproxy

4.2.1.2 Perancangan Monitoring Throughput dan Latency dari Cbench

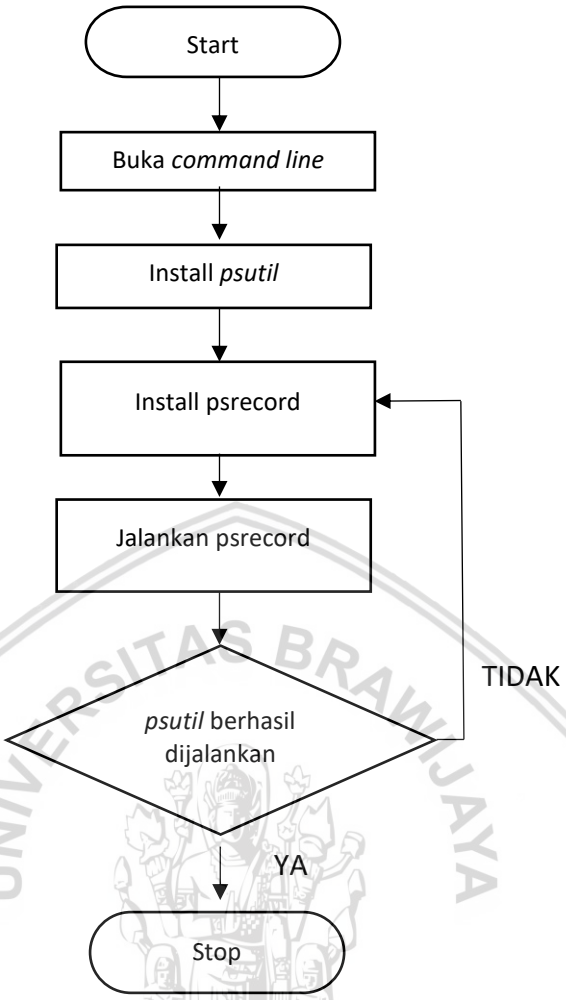
Perancangan monitoring *throughput* dan *latency* dari Cbench diperlukan untuk melakukan pengujian dalam penelitian ini. Simulator Cbench memiliki banyak peran untuk melakukan pengujian, dari simulator ini kita dapat memanipulasi jumlah host dan switch yang akan digunakan. Selain itu pada simulator Cbench juga diberikan 2 mode pilihan untuk melakukan pengujian yaitu *throughput mode* digunakan dalam pengujian *Flow Setup Rate,* sedangkan *latency mode* digunakan dalam pengujian Flow Setup Delay. Alur perancangan monitoring *throughput* dan *latency* dari HAproxy dapat dilihat pada **Gambar 4.3** berikut:



Gambar 4.3 Perancangan Monitoring Throughput dan Latency dari Cbench

4.2.1.3 Perancangan Monitoring Loadbalancing dari Psutil

Perancangan monitoring *loadbalancing* dari psutil ini diperlukan untuk mengambil data beban CPU dan memori ketika pengujian dilakukan. Untuk mendapatkan informasi tersebut maka digunakan psrecord dimana *library* dari psutil digunakan untuk mengambil informasi dari CPU dan memori yang sedang berjalan berupa log yang dapat disimpan sesuai dengan parameter yang kita masukkan. Alur perancangan monitoring *loadbalancing* dari Psutil dapat dilihat pada **Gambar 4.4** berikut:



Gambar 4.4 Perancangan Monitoring *Loadbalancing* dari PSUTIL

4.2.2 Perancangan Jaringan

Setelah selesai dengan perancangan perangkat lunak maka langkah selanjutnya yaitu dengan melakukan perancangan arsitektur jaringan dan koneksi kontroler dengan HAproxy. Perancangan arsitektur akan memberikan gambaran alamat tiap device beserta perannya dengan cara berkomunikasi menggunakan model client-server. Selanjutnya akan dilanjutkan dengan menghubungakan antara kontroler dengan HAproxy agar semua request yang masuk dapat diterima oleh kedua kontroler dengan beban yang seimbang menggunakan loadbalancing yang disediakan oleh HAproxy, maka dari itu dibutuhkan perancangan jaringan.

4.2.2.1 Perancangan Arsitektur Jaringan

Perancangan arsitektur jaringan dibutuhkan terkait dengan komunikasi antara mesin yang digunakan. Arsitektur jaringan yang digunakan adalah model komunikasi client-server, oleh karena itu dalam perancangan ini akan dijelaskan gambaran mesin yang digunakan pada client-server. Pada server memiliki 3 mesin yaitu:

- Virtual Machine server 1 berisi kontroler POX
- Virtual Machine server 2 berisi kontroler POX
- Virtual Machine server 3 berisi kontroler POX
- Host server berisi HAproxy

Sedangkan pada VM client memiliki satu mesin yaitu:

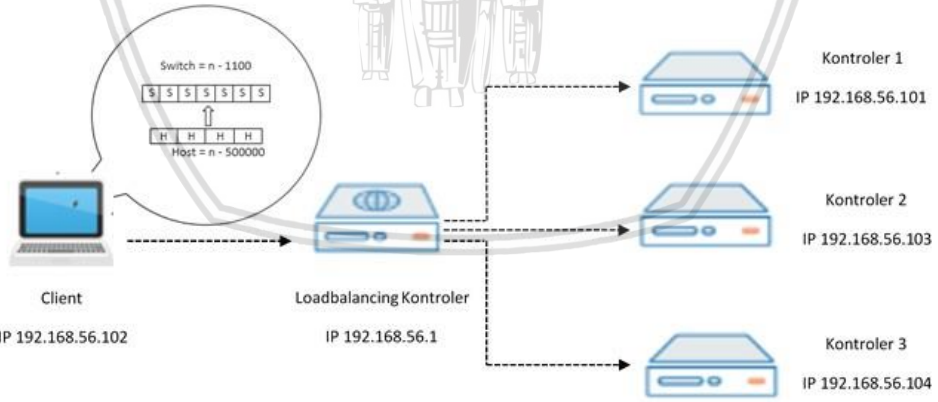
- Virtual Machine client berisi simulator Cbench

Pada perancangan arsitektur jaringan juga dilakukan konfigurasi pengalamatan IP dan pembagian peran pada setiap mesin dapat dilihat pada **Tabel 4.1** berikut:

Tabel 4.1 Pengalamatan IP

IP	Peran
192.168.56.101	Kontroler 1
192.168.56.103	Kontroler 2
192.168.56.104	Kontroler 3
192.168.56.1	*Loadbalancing server*
192.168.56.102	*Client*

Didalam sistem nantinya gambaran terkait dengan peranan dari masing-masing perangkat yang terhubung didalam sistem dapat dilihat pada **Gambar 4.5** berikut:

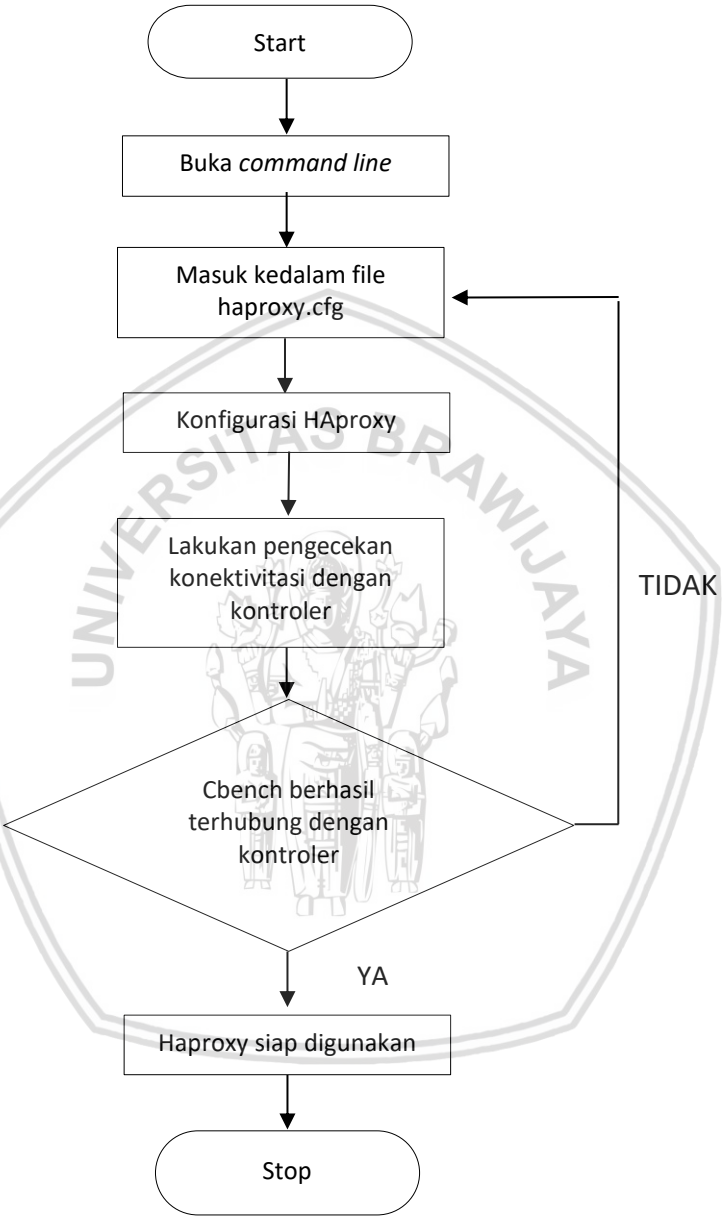


Gambar 4.5 Rancangan Peran Tiap Mesin dalam Sistem

4.2.2.2 Perancangan Koneksi Kontroler dengan HAproxy

Perancangan koneksi kontroler dengan HAproxy diperluakn untuk membuat keduanya terkoneksi sehingga semua request yang menuju ke kontroler akan dibagi secara merata oleh HAproxy dengan *loadbalancing.* Koneksi dalam perancangan ini diperlukan untuk melakukan pengujian loadbalancing yang berjalan untuk menyeimbangkan beban ketiga kontroler POX. Ketika keduanya

sudah terhubung maka HAproxy juga dapat melakukan berbagai monitoring terhadap ketiga kontroler POX yang ada, salah satunya yaitu menampilkan status kontroler POX yang terhubung apakah sedang aktif atau sebaliknya. Alur perancangan koneksi kontroler dengan HAproxy dapat dilihat pada **Gambar 4.6** berikut:



Gambar 4.6 Perancangan Koneksi Kontroler dengan HAproxy

BAB 5 IMPLEMENTASI

Bab ini membahas mengenai tahapan implementasi perangkat lunak dan implementasi jaringan untuk membentuk lingkungan pengujian yang nantinya akan didapatkan hasil untuk dianalisis pada bab selanjutnya. Dalam implementasi perangkat lunak terdapat konfigurasi konfigurasi HAproxy, konfigurasi simulator Cbench, dan konfigurasi Psutil. Sedangkan pada implementasi jaringan terdapat implementasi arsitektur dan implementasi koneksi kontroler dengan HAproxy.

5.1 Implementasi Perangkat Lunak

Implementasi perangkat lunak berisi tentang penjelasan bagaimana melakukan konfigurasi antara kontroler dan simulator pada *Virtual Machine*, selain itu juga mengatur konfigurasi HAproxy pada Host yang nantinya akan dapat menjalankan fungsi dari *loadbalancing.*

5.1.1 Implementasi Konfigurasi Loadbalancing dari HAproxy

Konfigurasi loadbalancing akan dilakukan melalui HAproxy, dimana HAproxy akan berada pada Host PC. HAproxy ini akan mendapatkan IP sesuai yang telah dikonfigurasi pada *Virtual Machine* dalam sub bab sebelumnya. Pertama yang dilakukan adalah menginstall HAproxy dengan memasukkan perintah berikut:

```
$ sudo apt-get install haproxy
```

Setelah install selesai, selanjutnya cek versi untuk mengetahui apakah haproxy sudah terinstall dengan benar dengan perintah:

```
$ haproxy -v
```

Hasil yang akan ditampilkan jika sudah terinstall dengan benar adalah seperti pada **Gambar 5.1** dibawah ini:

```
adi@user-1351502011110 98:~$ haproxy -v
HA-Proxy version 1.4.24 2013/06/17
Copyright 2000-2013 Willy Tarreau <w@1wt.eu>
```

Gambar 5.1 HAproxy version

Langkah selanjutnya membuat HAproxy berubah status menjadi ENABLED dengan memasukkan perintah:

```
$ sudo nano /etc/default/haproxy
```

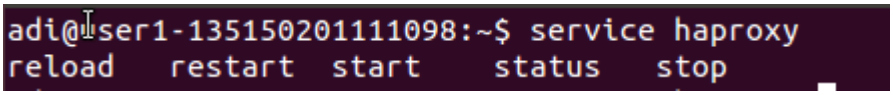
Opsi untuk merubah mejadi ENABLED adalah dengan mengganti nilai 0 menjadi 1 seperti script dibawah ini:

```
ENABLED=1
```

Jika sudah maka langkah selanjutnya adalah mencoba untuk menjalankan HAproxy dengan syntax seperti berikut:

```
$ service haproxy
```

HAproxy sudah siap digunakan jika sudah menampilkan status seperti **Gambar 5.2** dibawah ini:



Gambar 5.2 HAproxy status

5.1.2 Implementasi Monitoring Throughput dan Latency dari Cbench

Simulator Cbench akan dikonfigurasikan dalam Virtual Machine Client dimana bertugas untuk melakukan monitoring yang nantinya akan digunakan dalam pengujian baik throughput dan latency. Langkah-langkah dalam menginstal simulator Cbench adalah sebagai berikut:

- Instalasi dependensi yang diperlukan dengan memasukkan perintah

  ```
  $ sudo apt-get install autoconf automake libtool libsnmp-dev libpcap-dev libconfig8-dev
  ```

- Mengunduh berkas Openflow melalui website github dengan perintah

  ```
  $ git clone git://gitosis.stanford.edu/openflow.git
  ```

- Mengaktifkan versi source code OpenFlow dengan perintah

  ```
  $ cd openflow; git checkout –b mybranch origin/release/1.0.0
  ```

- Mengunduh paket Oflops pada website github dengan perintah

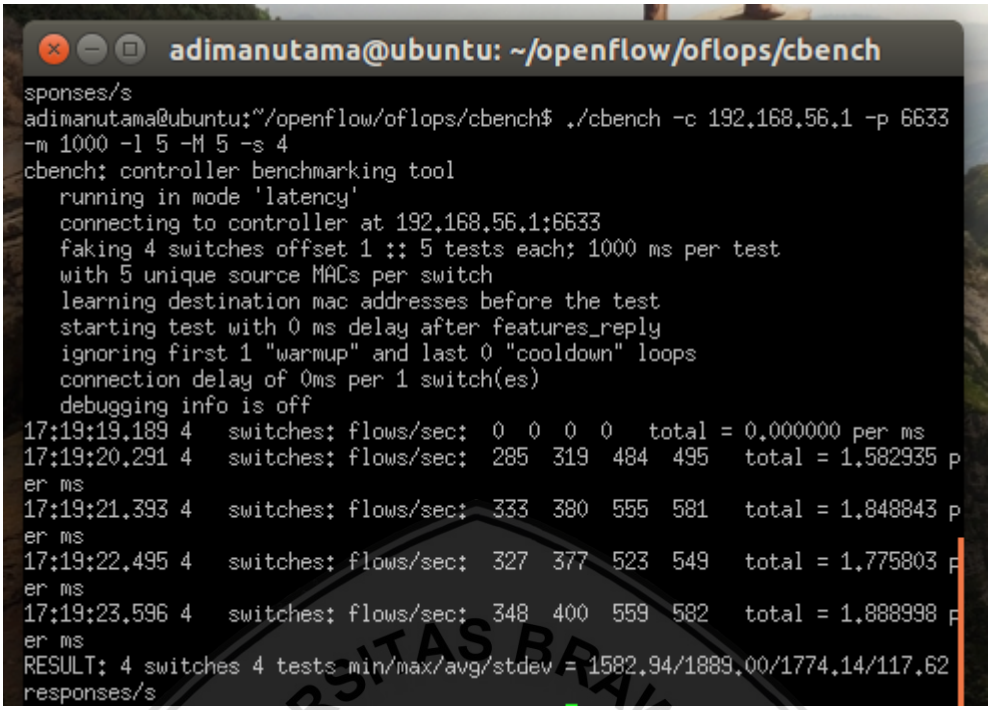
  ```
  $ git clone git://gitosis.stanford.edu/oflops.git
  ```

  Oflops merupakan singkatan dari OpenFlow Operations Per Second yaitu paket yang berisi OpenFlow Debugging tools dan Cbench.

- Membangun source code Oflops dengan perintah

  ```
  $ cd oflops; sh ./boot.sh; ./configure –with-openflow-src-dir=/home/Adimanutama/openflow; make; make install
  ```

Setelah instalasi selesai dilakukan maka simulator Cbench dapat digunakan dengan cara masuk terlebih dahulu ke direktori cbench dengan perintah $cd cbench. Seperti yang ditunjukkan pada **Gambar 5.3** simulator Cbench telah berhasil dijalankan pada terminal.



Gambar 5.3 Simulator Cbench

5.1.3 Implementasi Monitoring Loadbalancing dari Psutil

Psutil ini akan diimplementasikan pada server HAproxy, server 2 dan server 3 untuk dilkaukan monitoring ketika pengujian pada server tersebut dijalankan, sehingga akan didapatkan informasi berupa penggunaan CPU dan memori dari server tersebut. Langkah-langkah dalam melakukan dalam menginstal *library* psutil dan psrecord adalah sebagai berikut:

- Instal *dependencies* python

```
$ sudo apt-get install gcc python-dev python-pip
```

- Instal psutil

```
$ pip install psutil
```

- Install psrecord

```
$ pip install psrecord
```

Setelah semua selesai di install maka psrecord siap digunakan untuk mengambil data informasi dari CPU dan memori dengan memasukkan beberapa parameter seperti PID, nama file, interval, dan berapa lama durasi yang diperlukan selama satu kalo psrecord diaktifkan.

5.2 Implementasi Jaringan

Implementasi jaringan berisi tentang bagaimana komunikasi akan berjalan dalam satu jaringan dimana terdapat 3 kontroler yang berperan sebagai server, HAproxy yang berperan sebagai *load balancing* untuk ketiga kontroler tersebut dan 1 client yang berisi simulator Cbench untuk melakukan pengujian terhadap *load balancing* kontroler yang telah disiapkan.

5.2.1 Implementasi Arsitektur Jaringan

Arsitektur jaringan yang digunakan adalah model komunikasi client-server. Banyak server yang diperlukan dalam pengujian ini sejumlah 3 server yang dibangun pada *Virtual Machine* yang masing-masing berisi kontroler POX yang telah selesai dikonfigurasikan. Ketiga server ini nantinya akan terhubung dengan HAproxy sebagai *loadbalancing* server yang dikonfigurasikan pada host PC dan akan dijelaskan pada sub bab selanjutnya. Oleh karena itu konfigurasi pengalamatan IP diperlukan untuk memberikan identitas tiap server agar dapat diakses oleh satu sama lain. Pengalamantan IP akan diberikan sesuai dengan **Tabel 4.1** yang telah dituliskan pada bab perancangan. Langkah-langkah pengalamatan tiap mesin adalah sebagai berikut:

1. Virtual Machine server 1 berisi kontroler POX

   Setelah masuk kedalam mesin 1, ketikkan perintah seperti dibawah ini

   ```
   $ sudo nano /etc/network/interfaces
   ```

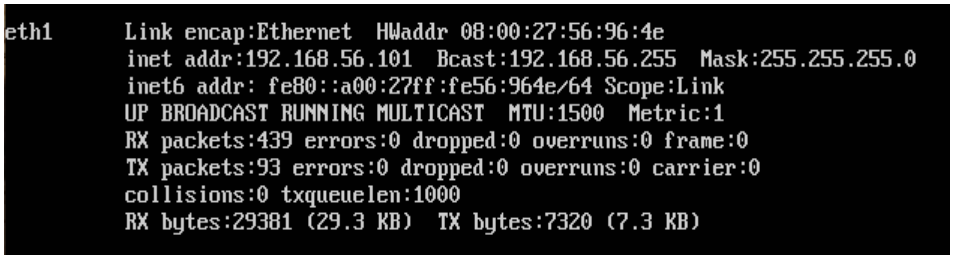
   Lalu tambahkan beberapa syntax didalamnya dengan memasukkan kode berikut

   ```
   auto eth1
   iface eth1 inet static
           address 192.168.56.101
           netmask 255.255.255.0
           broadcast 192.168.56.255
   ```

   Setelah selesai langkah selanjutnya melakukan restart network agar pengaturan yang sebelumnya telah dikonfigurasi dapat beroperasi, lakukan restarting dengan mengetikkan perintah seperti berikut:

   ```
   $ sudo /etc/init.d/networking restart
   ```

   Jika sudah selesai melakukan restart sekarang masukkan perintah $ifconfig dan lihat hasilnya, jika sudah seperti pada **Gambar 5.4** maka setting pengalamatan IP pada mesin 1 sudah berhasil.



Gambar 5.4 Konfigurasi IP VM kontroler 1

2. Virtual Machine server 2 berisi kontroler POX

   Setelah masuk kedalam mesin 2, ketikkan perintah seperti dibawah ini

```
$ sudo nano /etc/network/interfaces
```

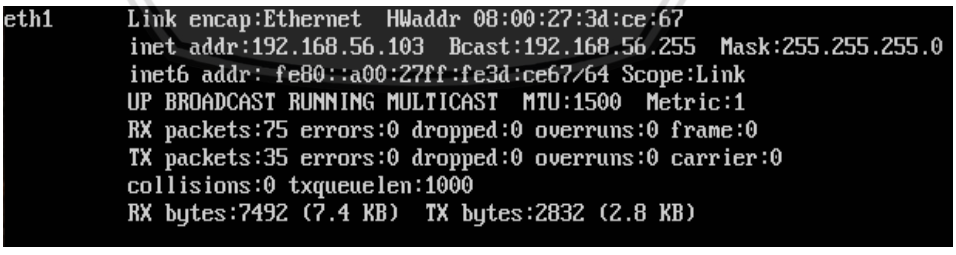
   Lalu tambahkan beberapa syntax didalamnya dengan memasukkan kode berikut

```
auto eth1
iface eth1 inet static
        address 192.168.56.103
        netmask 255.255.255.0
        broadcast 192.168.56.255
```

   Setelah selesai langkah selanjutnya melakukan restart network agar pengaturan yang sebelumnya telah dikonfigurasi dapat beroperasi, lakukan restarting dengan mengetikkan perintah seperti berikut:

```
$ sudo /etc/init.d/networking restart
```

   Jika sudah selesai melakukan restart sekarang masukkan perintah $ifconfig dan lihat hasilnya, jika sudah seperti pada **Gambar 5.5** maka setting pengalamatan IP pada mesin 2 sudah berhasil.



Gambar 5.5 Konfigurasi IP VM kontroler 2

3. Virtual Machine server 3 berisi kontroler POX

   Setelah masuk kedalam mesin 3, ketikkan perintah seperti dibawah ini

```
$ sudo nano /etc/network/interfaces
```

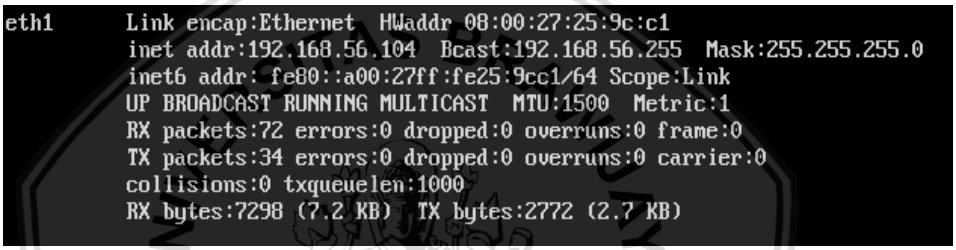
   Lalu tambahkan beberapa syntax didalamnya dengan memasukkan kode berikut

```
auto eth1
iface eth1 inet static
        address 192.168.56.104
        netmask 255.255.255.0
        broadcast 192.168.56.255
```

Setelah selesai langkah selanjutnya melakukan restart network agar pengaturan yang sebelumnya telah dikonfigurasi dapat beroperasi, lakukan restarting dengan mengetikkan perintah seperti berikut:

```
$ sudo /etc/init.d/networking restart
```

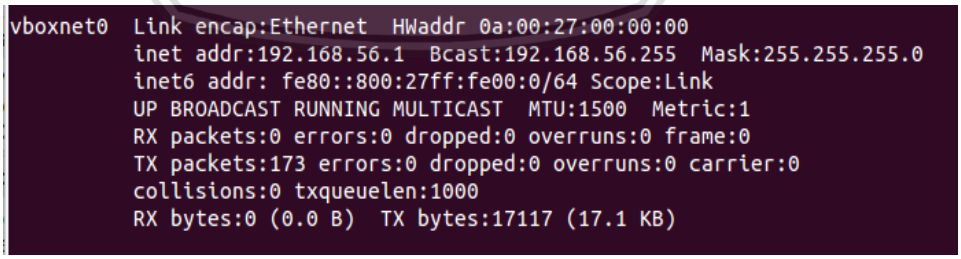
Jika sudah selesai melakukan restart sekarang masukkan perintah $ifconfig dan lihat hasilnya, jika sudah seperti pada **Gambar 5.6** maka setting pengalamatan IP pada mesin 3 sudah berhasil.



Gambar 5.6 Konfigurasi IP VM kontroler 3

4. Host server berisi HAproxy

   Pengaturan pengalamatan IP pada HAproxy ini sebenarnya sudah ditunjukkan pada bab sebelumnya pada konfigurasi *loadbalancing* dari HAproxy, karena pengaturan IP terdapat pada Virtual Box. Jika pengaturan sudah benar maka periksa alamat HAproxy itu melalui Host PC dengan memasukkan perintah $ifconfig. Jika sudah muncul seperti pada **Gambar 5.7** dibawah ini maka pengaturan sudah benar.



Gambar 5.7 Konfigurasi IP *load balancing sever* HAproxy

5. Virtual Machine client berisi simulator Cbench

   Setelah masuk kedalam mesin 4, ketikkan perintah seperti dibawah ini

```
$ sudo nano /etc/network/interfaces
```

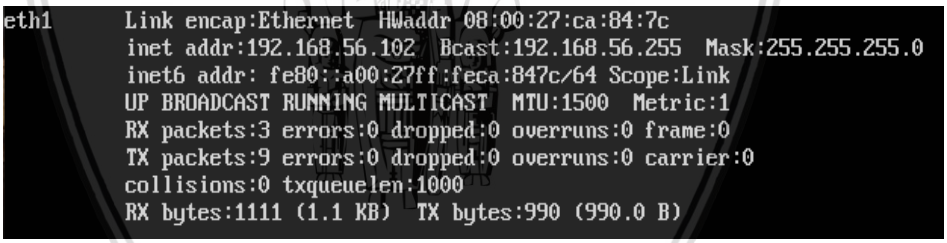
   Lalu tambahkan beberapa syntax didalamnya dengan memasukkan kode berikut

```
auto eth1
iface eth1 inet static
        address 192.168.56.102
        netmask 255.255.255.0
        broadcast 192.168.56.255
```

   Setelah selesai langkah selanjutnya melakukan restart network agar pengaturan yang sebelumnya telah dikonfigurasi dapat beroperasi, lakukan restarting dengan mengetikkan perintah seperti berikut:

```
$ sudo /etc/init.d/networking restart
```

   Jika sudah selesai melakukan restart sekarang masukkan perintah $ifconfig dan lihat hasilnya, jika sudah seperti pada **Gambar 5.8** maka setting pengalamatan IP pada mesin 4 sudah berhasil.



Gambar 5.8 Konfigurasi IP VM *client*

5.2.2 Implementasi Koneksi Kontroler dengan HAproxy

Setelah konfigurasi pengalamatan IP pada setiap mesin server dan mesin client sudah sesuai dengan perancangan yang dibuat maka langkah selanjutnya adalah membuat semua mesin server menjadi terkoneksi, dimana artinya adalah HAproxy sebagai *loadbalancing* server dapat terkoneksi dengan ketiga server yang nantinya akan memberikan pembagian beban secara merata ketika server mendapatkan request dari client. Langkah-langkah yang harus dilakukan untuk mengkoneksikan HAproxy dengan 3 server tersebut adalah sebagai berikut:

1. Pada Host PC ketikkan perintah berikut:

```
$ sudo nano /etc/haproxy/haproxy.cfg
```

2. Selanjutnya ubah beberapa code agar *loadbalancing* dapat terkoneksi dengan ketiga server kontroler sebagai berikut:

```
listen SDN 192.168.56.1:6633
        mode tcp
        balance roundrobin
        server POX_1 192.168.56.101:6633 check
        server POX_2 192.168.56.103:6633 check
        server POX_3 192.168.56.104:6633 check
listen stats 0.0.0.0:8080
        mode http
        stats enable
        stats uri /haproxy?stats
        balance roundrobin
        option httpclose
```

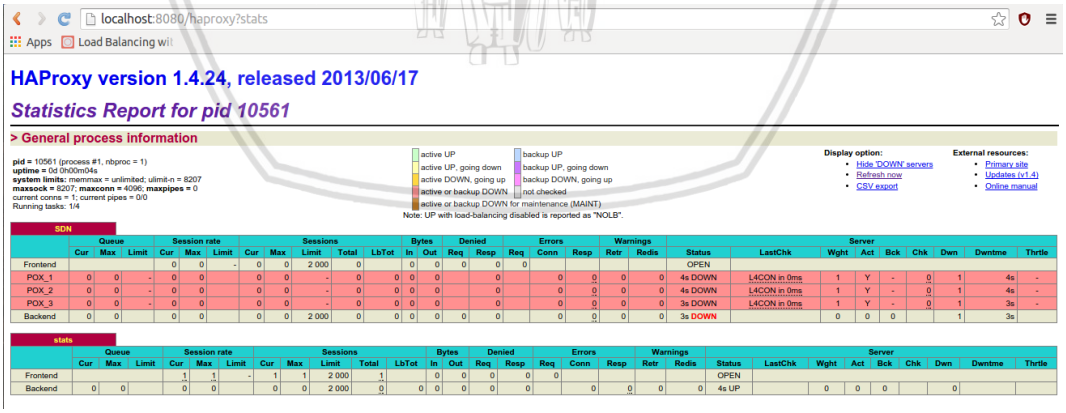
3. Setelah pengaturan disimpan, jalankan HAproxy untuk mengetahui status yang dapat dilihat pada browser. Jalankan HAproxy dengan perintah sebagai berikut:

```
$ sudo service haproxy start
```

4. Buka web browser dan masukkan alamat dalam kolom URL.

```
localhost:8080/haproxy?stats
```

5. Jika keluarannya sudah seperti pada **Gambar 5.9** berikut maka HAproxy sudah siap digunakan untuk melakukan loadbalancing pada ketiga server



Gambar 5.9 HAproxy status pada browser

BAB 6 PENGUJIAN DAN ANALISIS HASIL PENGUJIAN

Pada bab ini akan dijelaskan bagaimana pengujian akan dilakukan dan dianalisis hasilnya dari pengujian tersebut. Pengujian yang dilakukan juga mengacu dengan skenario pengujian yang telah dibuat pada bab sebelumnya yang meliputi pengujian *flow setup rate* dan pengujian *flow setup delay* dimana keduanya merupakan representasi dari *latency* dan *throughput*.

6.1 Pengujian

Terdapat 5 model pengujian yang akan dilakukan pada *loadbalancing* kontroler POX yaitu pengujian *flow setup delay* dan pengujian *flow setup rate*, dimana secara umum urutan pengujian yang akan dilakukan seperti pada **Tabel 6.1** berikut:

Tabel 6.1 Urutan Pengujian

No	Jumlah Switch dan Host	Pengujian
1	Switch Variasi dan Host Tetap	Uji Latency 1 Kontroler POX
		Uji Latency 2 kontroler POX
		Uji Latency 3 kontroler POX
2	Switch Tetap dan Host Variasi	Uji Latency 1 Kontroler POX
		Uji Latency 2 Kontroler POX
		Uji Latency 3 Kontroler POX
3	Switch Tetap dan Host Variasi	Uji Throughput 1 Kontroler POX
		Uji Throughput 2 Kontroler POX
		Uji Throughput 3 Kontroler POX
4	Switch Variasi dan Host Tetap	Uji Throughput 1 Kontroler POX
		Uji Throughput 2 Kontroler POX
		Uji Throughput 3 Kontroler POX
5	Switch Tetap dan Host Variasi	Uji CPU dan Memori usage

6.1.1 Pengujian Flow Setup Delay dengan Variasi Switch

Pengujian *Flow Setup Delay* atau *latency mode* ini akan dijalankan menggunakan simulator Cbench, dimana secara *default* simulator Cbench sudah berjalan pada *mode latency.* Tujuan dari pengujian ini adalah untuk mengetahui tingkat kecepatan respon per detik untuk setiap switch yang dikontrol yang dihasilkan dari masing-masing kontroler dengan jumlah kontroler antara 1 hingga 3 menggunakan *loadbalancing* dari HAproxy. Variable yang perlu dimasukkan

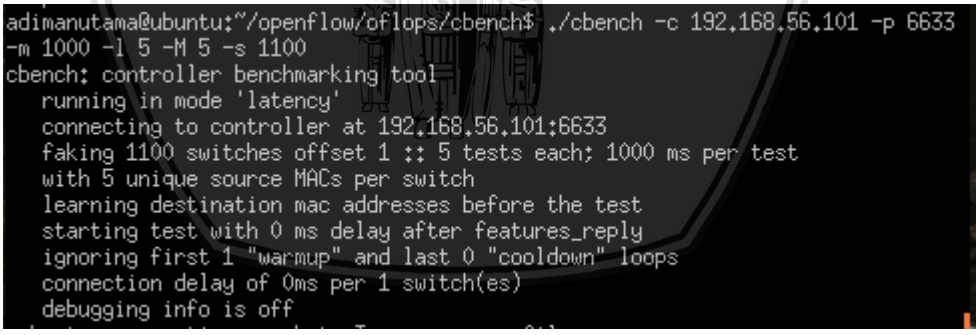
untuk melakukan pengujian latency pada simulator Cbench sesuai dengan skenario pengujian dapat dilihat pada **Tabel 6.2** berikut:

Tabel 6.2 Variabel dan Nilai pada Pengujian Latency dengan variasi switch

Uji Latency Kontroler POX	
Parameter	**Nilai**
IP (-c)	192.168.56.101/192.168.56.1
Port (-p)	6633
Waktu (-m)	1000
Iterasi (-l)	5
Host (-M)	5
Variasi Switch (-s)	100,200,300,400,500,600,700,800,900,1000.
Mode (-t)	Tidak Ditulis

6.1.1.1 Uji Latency 1 Kontroler POX

Pada pengujian *latency* 1 kontroler yang harus dilakukan pertama adalah mengaktifkan 1 kontroler POX, dengan menjalankan satu mesin *Virtual Machine* server maka kontroler siap digunakan. Setelah itu jalankan simulator Cbench dengan memasukkan variabel yang telah ditentukan pada skenario pengujian, berikut tampilan pengujian yang ditunjukkan dari terminal Cbench.

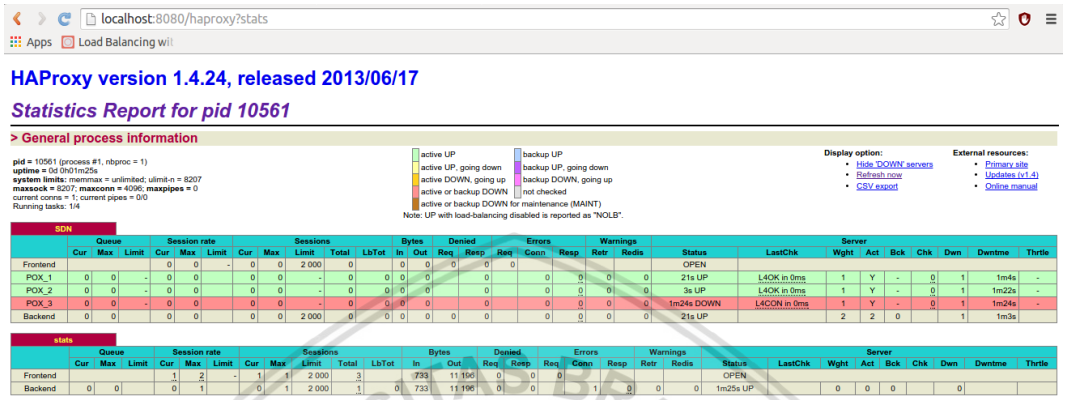


Gambar 6.1 Pengujian Flow Setup Delay 1 kontroler dengan variasi switch

Seperti yang ditunjukkan **Gambar 6.1** ada beberapa variabel yang harus dimasukkan untuk melakukan pengujian. Pertama adalah variabel –c yang harus diisi alamat IP dari kontroler, berhubung pengujian ini hanya menggunakan 1 server maka IP yang dituliskan adalah IP dari kontroler 1 itu sendiri (tanpa melalui HAproxy), selanjutnya variabel –p adalah port yang digunakan oleh kontroler POX, lalu variabel –m merupakan waktu yang diperlukan dalam satu kali iterasi dengan satuan *miliseconds*. Variablel –l merupakan banyak iterasi yang dilakukan dalam satu kali pengujian berjalan, lalu variabel –M merupakan banyak host dan variabel –s merupakan banyak switch yang akan digunakan sesuai dengan skenario yang telah dibuat.

6.1.1.2 Uji Latency 2 Kontroler POX

Pada pengujian *latency* 2 kontroler yang harus dilakukan pertama adalah mengaktifkan HAproxy dan 2 kontroler POX, dengan menjalankan dua mesin *Virtual Machine* server maka kontroler siap digunakan. Setelah itu cek status pada *web browser* apakah kontroler sudah siap digunakan seperti pada **Gambar 6.2** berikut:



Gambar 6.2 HAproxy status 2 kontroler aktif

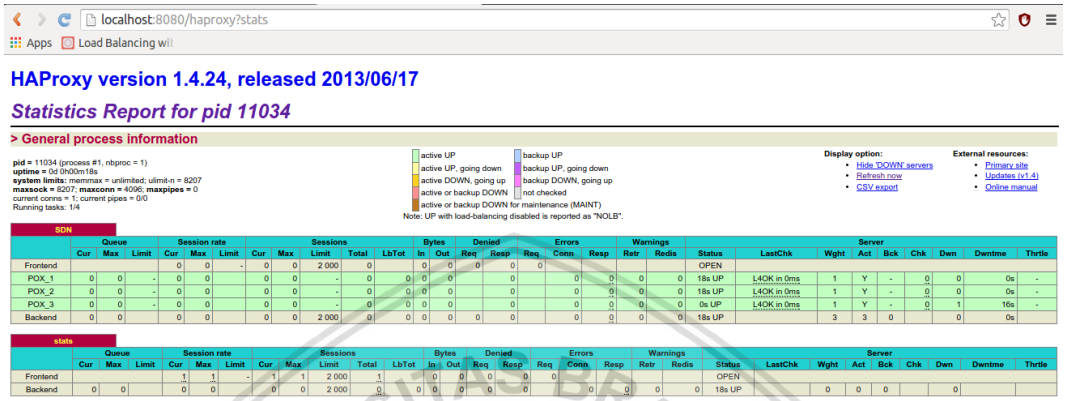
Selanjutnya adalah menjalankan simulator Cbench dengan memasukkan variabel yang telah ditentukan pada skenario pengujian, berikut tampilan pengujian yang ditunjukkan dari terminal Cbench.

Gambar 6.3 Pengujian Flow Setup Delay 2 kontroler dengan variasi switch

Seperti yang ditunjukkan **Gambar 6.3** ada beberapa variabel yang harus dimasukkan untuk melakukan pengujian. Pertama adalah variabel –c yang harus diisi alamat IP dari server *loadbalancing*, selanjutnya variabel –p adalah port yang digunakan oleh kontroler POX, lalu variabel –m merupakan waktu yang diperlukan dalam satu kali iterasi dengan satuan *miliseconds*. Variabel –l merupakan banyak iterasi yang dilakukan dalam satu kali pengujian berjalan, lalu variabel –M merupakan banyak host dan variabel –s merupakan banyak switch yang akan digunakan sesuai dengan skenario yang telah dibuat.

6.1.1.3 Uji Latency 3 Kontroler POX

Pada pengujian *latency* 3 kontroler yang harus dilakukan pertama adalah mengaktifkan HAproxy dan 3 kontroler POX, dengan menjalankan tiga mesin *Virtual Machine* server maka kontroler siap digunakan. Setelah itu cek status pada *web browser* apakah kontroler sudah siap digunakan seperti pada **Gambar 6.4** berikut:



Gambar 6.4 HAproxy status 3 kontroler aktif

Selanjutnya adalah menjalankan simulator Cbench dengan memasukkan variabel yang telah ditentukan pada skenario pengujian, berikut tampilan pengujian yang ditunjukkan dari terminal Cbench.

```
adimanutama@ubuntu:~/openflow/oflops/cbench$ ./cbench -c 192.168.56.1 -p 6633 -m
000 -l 5 -M 5 -s 1100
cbench: controller benchmarking tool
   running in mode 'latency'
   connecting to controller at 192.168.56.1:6633
   faking 1100 switches offset 1 :: 5 tests each; 1000 ms per test
   with 5 unique source MACs per switch
   learning destination mac addresses before the test
   starting test with 0 ms delay after features_reply
   ignoring first 1 "warmup" and last 0 "cooldown" loops
   connection delay of 0ms per 1 switch(es)
   debugging info is off
```

Gambar 6.5 Pengujian Flow Setup Delay 3 kontroler dengan variasi switch

Seperti yang ditunjukkan **Gambar 6.5** ada beberapa variabel yang harus dimasukkan untuk melakukan pengujian. Pertama adalah variabel –c yang harus diisi alamat IP dari server *loadbalancing*, selanjutnya variabel –p adalah port yang digunakan oleh kontroler POX, lalu variabel –m merupakan waktu yang diperlukan dalam satu kali iterasi dengan satuan *miliseconds*. Variabel –l merupakan banyak iterasi yang dilakukan dalam satu kali pengujian berjalan, lalu variabel –M merupakan banyak host dan variabel –s merupakan banyak switch yang akan digunakan sesuai dengan skenario yang telah dibuat.

6.1.2 Pengujian Flow Setup Delay dengan Variasi Host

Pengujian *Flow Setup Delay* atau *latency mode* ini akan dijalankan menggunakan simulator Cbench, dimana secara *default* simulator Cbench sudah berjalan pada *mode latency.* Tujuan dari pengujian ini adalah untuk mengetahui tingkat kecepatan respon per detik untuk setiap switch yang dikontrol yang dihasilkan dari masing-masing kontroler dengan jumlah kontroler antara 1 hingga 3 menggunakan *loadbalancing* dari HAproxy. Variable yang perlu dimasukkan untuk melakukan pengujian latency pada simulator Cbench sesuai dengan skenario pengujian dapat dilihat pada **Tabel 6.3** berikut:

Tabel 6.3 Variabel dan Nilai pada Pengujian Latency dengan variasi host

Uji Latency Kontroler POX	
Parameter	**Nilai**
IP (-c)	192.168.56.101/192.168.56.1
Port (-p)	6633
Waktu (-m)	1000
Iterasi (-l)	5
Variasi Host (-M)	2000,4000,6000,8000,10000.
Switch (-s)	15
Mode (-t)	Tidak Ditulis

6.1.2.1 Uji Latency 1 Kontroler POX

Pada pengujian *latency* 1 kontroler yang harus dilakukan pertama adalah mengaktifkan 1 kontroler POX, dengan menjalankan satu mesin *Virtual Machine* server maka kontroler siap digunakan. Setelah itu jalankan simulator Cbench dengan memasukkan variabel yang telah ditentukan pada skenario pengujian, berikut tampilan pengujian yang ditunjukkan dari terminal Cbench.

```
adimanutama@ubuntu:~/openflow/oflops/cbench$ ./cbench -c 192.168.56.101 -p 6633
-m 1000 -l 5 -M 7000 -s 15
cbench: controller benchmarking tool
   running in mode 'latency'
   connecting to controller at 192.168.56.101:6633
```

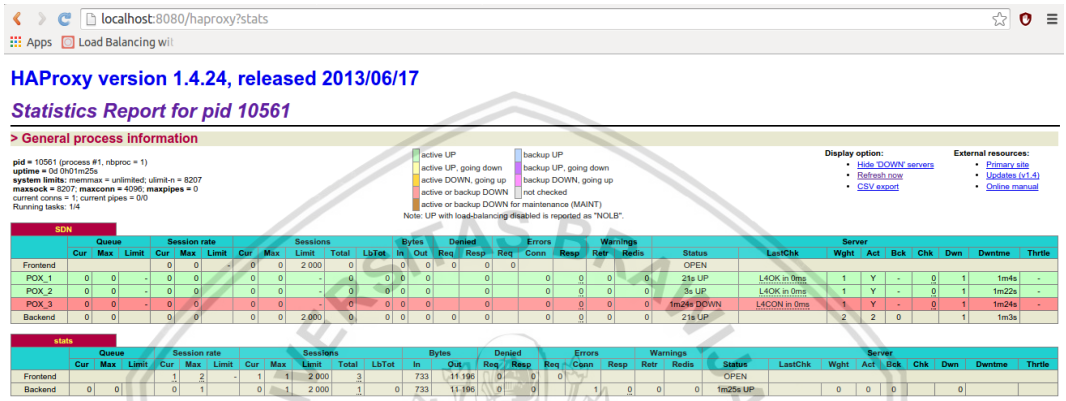
Gambar 6.6 Pengujian Flow Setup Delay 1 kontroler dengan variasi host

Seperti yang ditunjukkan **Gambar 6.6** ada beberapa variabel yang harus dimasukkan untuk melakukan pengujian. Pertama adalah variabel –c yang harus diisi alamat IP dari kontroler, berhubung pengujian ini hanya menggunakan 1 server maka IP yang dituliskan adalah IP dari kontroler 1 itu sendiri (tanpa melalui HAproxy), selanjutnya variabel –p adalah port yang digunakan oleh kontroler POX, lalu variabel –m merupakan waktu yang diperlukan dalam satu kali iterasi dengan satuan *miliseconds*. Variablel –l merupakan banyak iterasi yang dilakukan dalam

satu kali pengujian berjalan, lalu variabel –M merupakan banyak host dan variabel –s merupakan banyak switch yang akan digunakan sesuai dengan skenario yang telah dibuat.

6.1.2.2 Uji Latency 2 Kontroler POX

Pada pengujian *latency* 2 kontroler yang harus dilakukan pertama adalah mengaktifkan HAproxy dan 2 kontroler POX, dengan menjalankan dua mesin *Virtual Machine* server maka kontroler siap digunakan. Setelah itu cek status pada *web browser* apakah kontroler sudah siap digunakan seperti pada **Gambar 6.7** berikut:



Gambar 6.7 HAproxy status 2 kontroler aktif

Selanjutnya adalah menjalankan simulator Cbench dengan memasukkan variabel yang telah ditentukan pada skenario pengujian, berikut tampilan pengujian yang ditunjukkan dari terminal Cbench.

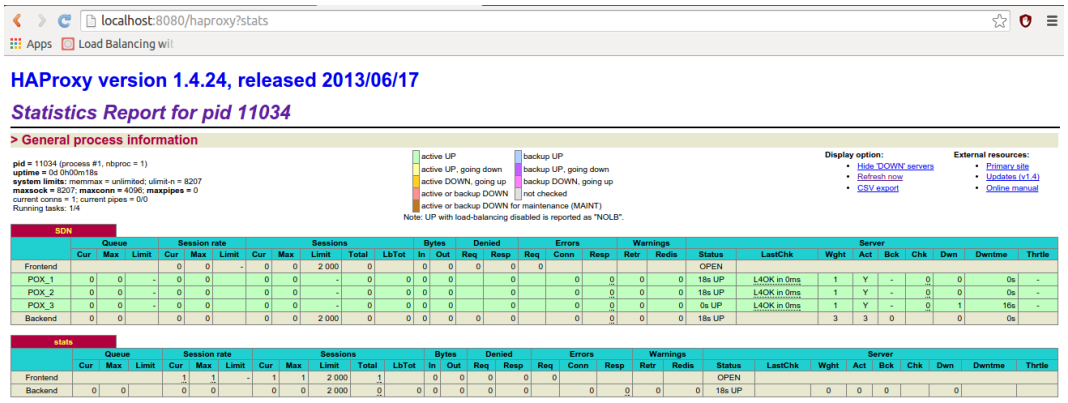
Gambar 6.8 Pengujian Flow Setup Delay 2 kontroler dengan variasi host

Seperti yang ditunjukkan **Gambar 6.8** ada beberapa variabel yang harus dimasukkan untuk melakukan pengujian. Pertama adalah variabel –c yang harus diisi alamat IP dari server *loadbalancing*, selanjutnya variabel –p adalah port yang digunakan oleh kontroler POX, lalu variabel –m merupakan waktu yang diperlukan dalam satu kali iterasi dengan satuan *miliseconds*. Variabel –l merupakan banyak iterasi yang dilakukan dalam satu kali pengujian berjalan, lalu variabel –M merupakan banyak host dan variabel –s merupakan banyak switch yang akan digunakan sesuai dengan skenario yang telah dibuat.

6.1.2.3 Uji Latency 3 Kontroler POX

Pada pengujian *latency* 3 kontroler yang harus dilakukan pertama adalah mengaktifkan HAproxy dan 3 kontroler POX, dengan menjalankan tiga mesin *Virtual Machine* server maka kontroler siap digunakan. Setelah itu cek status pada

web browser apakah kontroler sudah siap digunakan seperti pada **Gambar 6.9** berikut:



Gambar 6.9 HAproxy status 3 kontroler aktif

Selanjutnya adalah menjalankan simulator Cbench dengan memasukkan variabel yang telah ditentukan pada skenario pengujian, berikut tampilan pengujian yang ditunjukkan dari terminal Cbench.

Gambar 6.10 Pengujian Flow Setup Delay 3 kontroler dengan variasi host

Seperti yang ditunjukkan **Gambar 6.10** ada beberapa variabel yang harus dimasukkan untuk melakukan pengujian. Pertama adalah variabel –c yang harus diisi alamat IP dari server *loadbalancing*, selanjutnya variabel –p adalah port yang digunakan oleh kontroler POX, lalu variabel –m merupakan waktu yang diperlukan dalam satu kali iterasi dengan satuan *miliseconds*. Variabel –l merupakan banyak iterasi yang dilakukan dalam satu kali pengujian berjalan, lalu variabel –M merupakan banyak host dan variabel –s merupakan banyak switch yang akan digunakan sesuai dengan skenario yang telah dibuat.

6.1.3 Pengujian Flow Setup Rate dengan Variasi Host

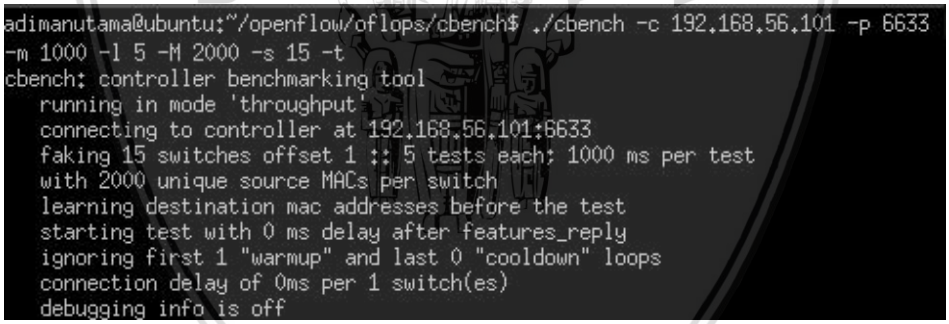
Pengujian *Flow Setup Rate* atau *throughput mode* ini akan dijalankan menggunakan simulator Cbench, dimana dengan menambah variabel –t pada simulator Cbench agar berjalan pada *mode throughput.* Tujuan dari pengujian ini adalah untuk mengetahui berapa banyak jumlah flow yang dapat ditangani berdasarkan besar arus data (flow) yang dapat dikelola oleh masing-masing kontroler setiap detiknya untuk setiap host yang dikelola. Variable yang perlu dimasukkan untuk melakukan pengujian throughput pada simulator Cbench sesuai dengan skenario pengujian dapat dilihat pada **Tabel 6.4** berikut:

Tabel 6.4 Variabel dan Nilai pada Pengujian Throughput dengan variasi host

Uji Throughput Kontroler POX	
Parameter	**Nilai**
IP (-c)	192.168.56.101/192.168.56.1
Port (-p)	6633
Waktu (-m)	1000
Iterasi (-l)	5
Variasi Host (-M)	2000,4000,6000,8000,10000.
Switch (-s)	15
Mode (-t)	Ditulis

6.1.3.1 Uji Throughput 1 Kontroler POX

Pada pengujian *throughput* 1 kontroler yang harus dilakukan pertama adalah mengaktifkan 1 kontroler POX, dengan menjalankan satu mesin *Virtual Machine* server maka kontroler siap digunakan. Setelah itu jalankan simulator Cbench dengan memasukkan variabel yang telah ditentukan pada skenario pengujian, berikut tampilan pengujian yang ditunjukkan dari terminal Cbench.

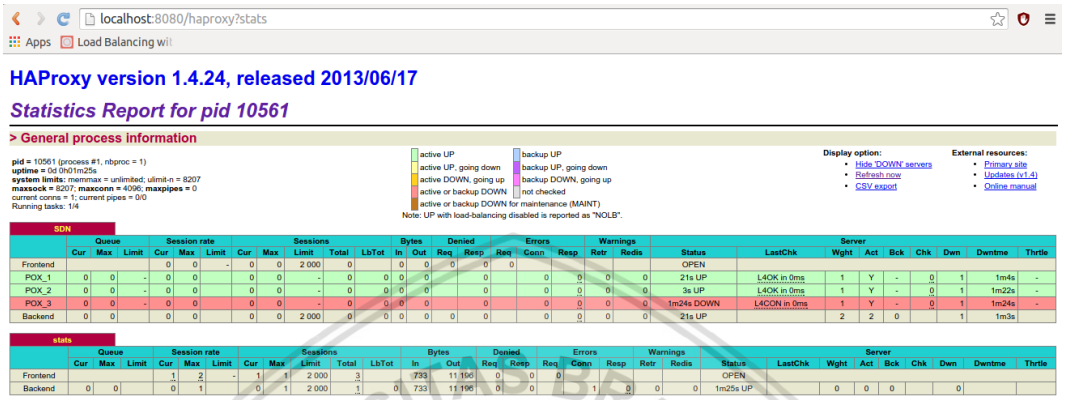


Gambar 6.11 Pengujian Flow Setup Rate 1 kontroler dengan variasi host

Seperti yang ditunjukkan **Gambar 6.11** ada beberapa variabel yang harus dimasukkan untuk melakukan pengujian. Pertama adalah variabel –c yang harus diisi alamat IP dari kontroler, berhubung pengujian ini hanya menggunakan 1 server maka IP yang dituliskan adalah IP dari kontroler 1 itu sendiri (tanpa melalui HAproxy), selanjutnya variabel –p adalah port yang digunakan oleh kontroler POX, lalu variabel –m merupakan waktu yang diperlukan dalam satu kali iterasi dengan satuan *miliseconds*. Variabel –l merupakan banyak iterasi yang dilakukan dalam satu kali pengujian berjalan, lalu variabel –M merupakan banyak host dan variabel –s merupakan banyak switch yang akan digunakan sesuai dengan skenario yang telah dibuat, dan terakhir adalah variabel –t yang akan merubah mode pengujian menjadi throughput.

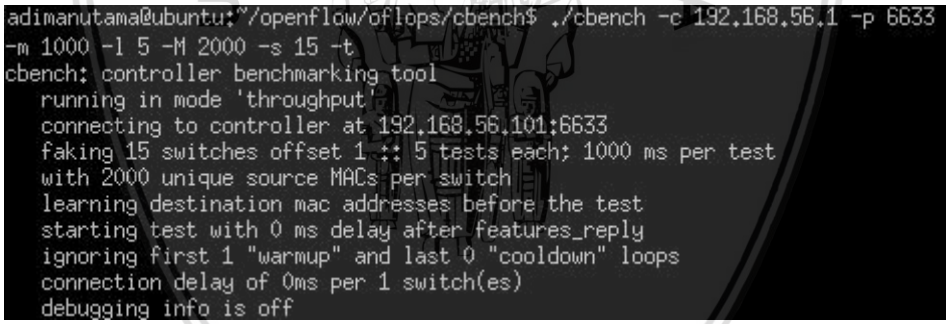
6.1.3.2 Uji Throughput 2 Kontroler POX

Pada pengujian *throughput* 2 kontroler yang harus dilakukan pertama adalah mengaktifkan HAproxy dan 2 kontroler POX, dengan menjalankan dua mesin *Virtual Machine* server maka kontroler siap digunakan. Setelah itu cek status pada *web browser* apakah kontroler sudah siap digunakan seperti pada **Gambar 6.12** berikut:



Gambar 6.12 HAproxy status 2 kontroler aktif

Selanjutnya adalah menjalankan simulator Cbench dengan memasukkan variabel yang telah ditentukan pada skenario pengujian, berikut tampilan pengujian yang ditunjukkan dari terminal Cbench.



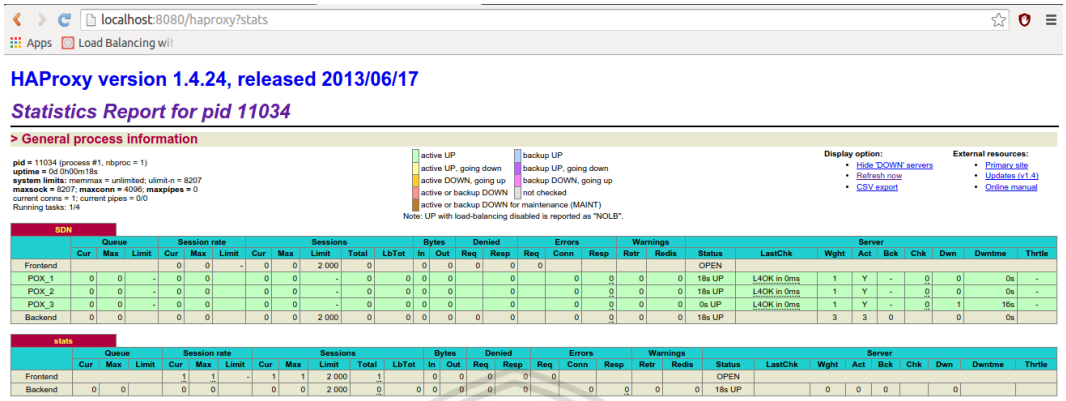
Gambar 6.13 Pengujian Flow Setup Rate 2 kontroler dengan variasi host

Seperti yang ditunjukkan **Gambar 6.13** ada beberapa variabel yang harus dimasukkan untuk melakukan pengujian. Pertama adalah variabel –c yang harus diisi alamat IP dari server *loadbalancing*, selanjutnya variabel –p adalah port yang digunakan oleh kontroler POX, lalu variabel –m merupakan waktu yang diperlukan dalam satu kali iterasi dengan satuan *miliseconds*. Variabel –l merupakan banyak iterasi yang dilakukan dalam satu kali pengujian berjalan, lalu variabel –M merupakan banyak host dan variabel –s merupakan banyak switch yang akan digunakan sesuai dengan skenario yang telah dibuat, dan terakhir adalah variabel –t yang akan merubah mode pengujian menjadi throughput.

6.1.3.3 Uji Throughput 3 Kontroler POX

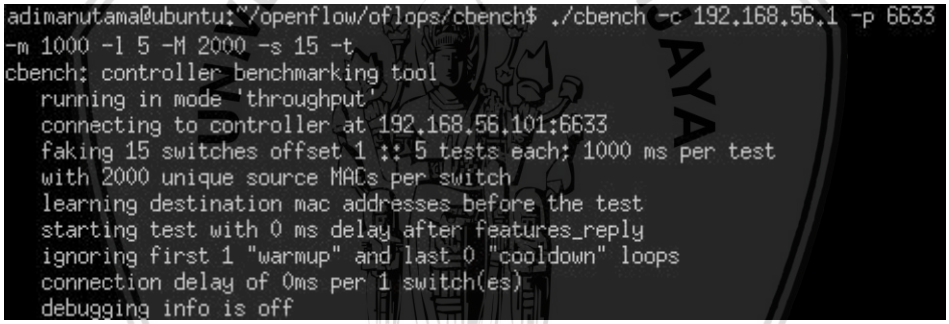
Pada pengujian *throughput* 3 kontroler yang harus dilakukan pertama adalah mengaktifkan HAproxy dan 3 kontroler POX, dengan menjalankan tiga

mesin *Virtual Machine* server maka kontroler siap digunakan. Setelah itu cek status pada *web browser* apakah kontroler sudah siap digunakan seperti pada **Gambar 6.14** berikut:



Gambar 6.14 HAproxy status 3 kontroler aktif

Selanjutnya adalah menjalankan simulator Cbench dengan memasukkan variabel yang telah ditentukan pada skenario pengujian, berikut tampilan pengujian yang ditunjukkan dari terminal Cbench.



Gambar 6.15 Pengujian Flow Setup Rate 3 kontroler dengan variasi host

Seperti yang ditunjukkan **Gambar 6.15** ada beberapa variabel yang harus dimasukkan untuk melakukan pengujian. Pertama adalah variabel –c yang harus diisi alamat IP dari server *loadbalancing*, selanjutnya variabel –p adalah port yang digunakan oleh kontroler POX, lalu variabel –m merupakan waktu yang diperlukan dalam satu kali iterasi dengan satuan *miliseconds*. Variabel –l merupakan banyak iterasi yang dilakukan dalam satu kali pengujian berjalan, lalu variabel –M merupakan banyak host dan variabel –s merupakan banyak switch yang akan digunakan sesuai dengan skenario yang telah dibuat, dan terakhir adalah variabel –t yang akan merubah mode pengujian menjadi throughput.

6.1.4 Pengujian Flow Setup Rate dengan Variasi Switch

Pengujian *Flow Setup Rate* atau *throughput mode* ini akan dijalankan menggunakan simulator Cbench, dimana dengan menambah variabel –t pada simulator Cbench agar berjalan pada *mode throughput.* Tujuan dari pengujian ini adalah untuk mengetahui berapa banyak jumlah flow yang dapat ditangani berdasarkan besar arus data (flow) yang dapat dikelola oleh masing-masing

kontroler setiap detiknya untuk setiap host yang dikelola. Variable yang perlu dimasukkan untuk melakukan pengujian throughput pada simulator Cbench sesuai dengan skenario pengujian dapat dilihat pada **tabel 6.5** berikut:

Tabel 6.5 Variabel dan Nilai pada Pengujian Throughput dengan variasi switch

Uji Throughput Kontroler POX	
Parameter	**Nilai**
IP (-c)	192.168.56.101/192.168.56.1
Port (-p)	6633
Waktu (-m)	1000
Iterasi (-l)	5
Host (-M)	5
Variasi Switch (-s)	100,200,300,400,500,600,700,800,900,1000.
Mode (-t)	Ditulis

6.1.4.1 Uji Throughput 1 Kontroler POX

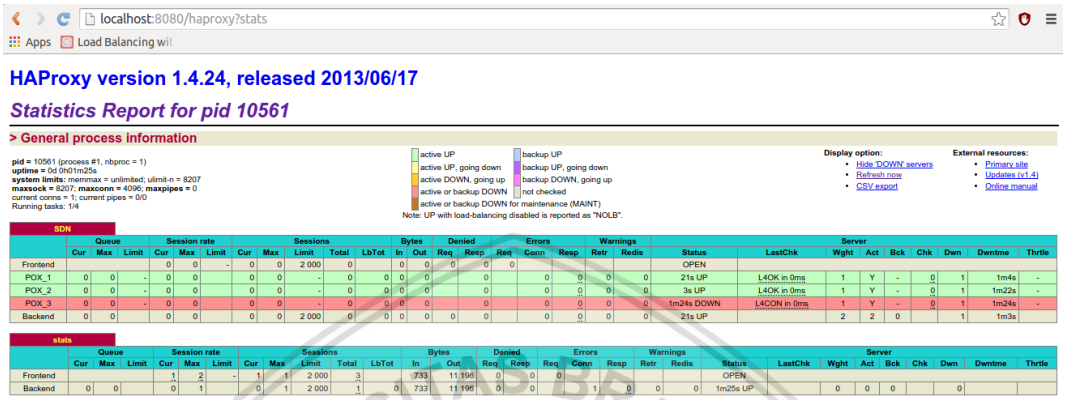
Pada pengujian *throughput* 1 kontroler yang harus dilakukan pertama adalah mengaktifkan 1 kontroler POX, dengan menjalankan satu mesin *Virtual Machine* server maka kontroler siap digunakan. Setelah itu jalankan simulator Cbench dengan memasukkan variabel yang telah ditentukan pada skenario pengujian, berikut tampilan pengujian yang ditunjukkan dari terminal Cbench.

Gambar 6.16 Pengujian Flow Setup Rate 1 kontroler dengan variasi switch

Seperti yang ditunjukkan **Gambar 6.16** ada beberapa variabel yang harus dimasukkan untuk melakukan pengujian. Pertama adalah variabel –c yang harus diisi alamat IP dari kontroler, berhubung pengujian ini hanya menggunakan 1 server maka IP yang dituliskan adalah IP dari kontroler 1 itu sendiri (tanpa melalui HAproxy), selanjutnya variabel –p adalah port yang digunakan oleh kontroler POX, lalu variabel –m merupakan waktu yang diperlukan dalam satu kali iterasi dengan satuan *miliseconds*. Variablel –l merupakan banyak iterasi yang dilakukan dalam satu kali pengujian berjalan, lalu variabel –M merupakan banyak host dan variabel –s merupakan banyak switch yang akan digunakan sesuai dengan skenario yang telah dibuat, dan terakhir adalah variabel –t yang akan merubah mode pengujian menjadi throughput.

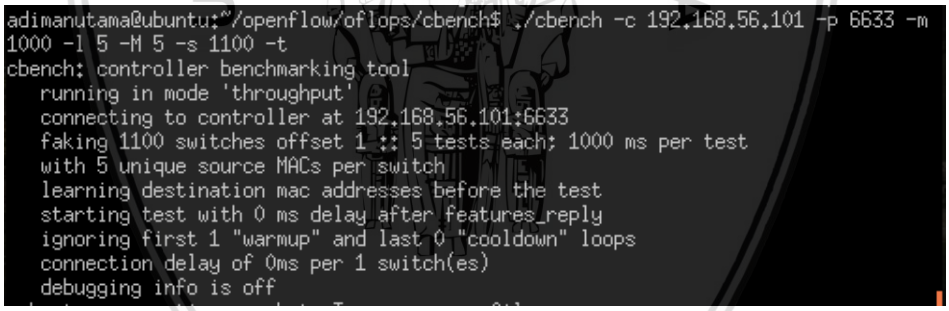
6.1.4.2 Uji Throughput 2 Kontroler POX

Pada pengujian *throughput* 2 kontroler yang harus dilakukan pertama adalah mengaktifkan HAproxy dan 2 kontroler POX, dengan menjalankan dua mesin *Virtual Machine* server maka kontroler siap digunakan. Setelah itu cek status pada *web browser* apakah kontroler sudah siap digunakan seperti pada **Gambar 6.17** berikut:



Gambar 6.17 HAproxy status 2 kontroler aktif

Selanjutnya adalah menjalankan simulator Cbench dengan memasukkan variabel yang telah ditentukan pada skenario pengujian, berikut tampilan pengujian yang ditunjukkan dari terminal Cbench.



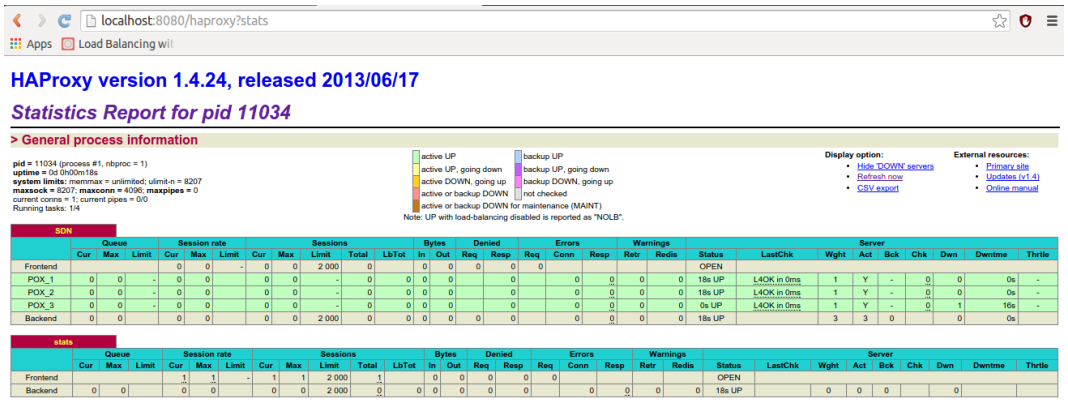
Gambar 6.18 Pengujian Flow Setup Rate 2 kontroler dengan variasi switch

Seperti yang ditunjukkan **Gambar 6.18** ada beberapa variabel yang harus dimasukkan untuk melakukan pengujian. Pertama adalah variabel –c yang harus diisi alamat IP dari server *loadbalancing*, selanjutnya variabel –p adalah port yang digunakan oleh kontroler POX, lalu variabel –m merupakan waktu yang diperlukan dalam satu kali iterasi dengan satuan *miliseconds*. Variablel –l merupakan banyak iterasi yang dilakukan dalam satu kali pengujian berjalan, lalu variabel –M merupakan banyak host dan variabel –s merupakan banyak switch yang akan digunakan sesuai dengan skenario yang telah dibuat, dan terakhir adalah variabel –t yang akan merubah mode pengujian menjadi throughput.

6.1.4.3 Uji Throughput 3 Kontroler POX

Pada pengujian *throughput* 3 kontroler yang harus dilakukan pertama adalah mengaktifkan HAproxy dan 3 kontroler POX, dengan menjalankan tiga mesin *Virtual Machine* server maka kontroler siap digunakan. Setelah itu cek

status pada *web browser* apakah kontroler sudah siap digunakan seperti pada **Gambar 6.19** berikut:



Gambar 6.19 HAproxy status 3 kontroler aktif

Selanjutnya adalah menjalankan simulator Cbench dengan memasukkan variabel yang telah ditentukan pada skenario pengujian, berikut tampilan pengujian yang ditunjukkan dari terminal Cbench.

Gambar 6.20 Pengujian Flow Setup Rate 3 kontroler dengan variasi switch

Seperti yang ditunjukkan **Gambar 6.20** ada beberapa variabel yang harus dimasukkan untuk melakukan pengujian. Pertama adalah variabel –c yang harus diisi alamat IP dari server *loadbalancing*, selanjutnya variabel –p adalah port yang digunakan oleh kontroler POX, lalu variabel –m merupakan waktu yang diperlukan dalam satu kali iterasi dengan satuan *miliseconds*. Variablel –l merupakan banyak iterasi yang dilakukan dalam satu kali pengujian berjalan, lalu variabel –M merupakan banyak host dan variabel –s merupakan banyak switch yang akan digunakan sesuai dengan skenario yang telah dibuat, dan terakhir adalah variabel –t yang akan merubah mode pengujian menjadi throughput.

6.1.5 Pengujian CPU dan Memori *Usage*

Pengujian CPU dan Memori *usage* ini akan dijalankan menggunakan library psutil yang dijalankan oleh psrecord, dimana psrecord akan mengambil informasi penggunaan berupa log sesuai dengan variabel yang digunakan. Tujuan dari pengujian ini adalah untuk mengetahui seberapa besar penggunaan CPU dan Memori pada tiap variasi jumlah kontroler ketika lingkungan sistem dijalankan untuk melakukan *loadbalancing.* Variable yang perlu dimasukkan untuk

melakukan pengujian CPU dan Memori *usage* oleh psrecord sesuai dengan skenario pengujian dapat dilihat pada **Tabel 6.5** berikut:

Tabel 6.6 Variabel dan Nilai pada Pengujian CPU dan Memori *Usage*

CPU dan Memori *Usage* server	
Parameter	**Nilai**
PID	PID HAproxy dan PID POX
File	nama_file.txt
Durasi	15
Interval	1
Variasi Host (-M)	2000,4000,6000,8000,10000.
Switch (-s)	15
Mode (-t)	Tidak Ditulis

Pengujian CPU dan Memori *Usage* ini dapat dilakukan bersamaan ketika melakukan pengujian *flow setup rate* dan *flow setup delay*. Ketika pengujian dijalankan maka yang perlu ditambah adalah terminal baru dari putty untuk masing-masing mesin kontroler yang sedang berjalan dan dengan bersamaan psrecord dijalankan untuk mengambil informasi dari masing-masing kontroler tersebut.

6.1.5.1 Uji CPU dan Memori *Usage* 1 Kontroler POX

Pada pengujian CPU dan Memori *Usage* dengan 1 kontroler ini, psrecord dijalankan dengan memasukkan variabel yang telah ditentukan pada skenario pengujian. Berikut tampilan pengujian yang ditunjukkan pada terminal:

```
$ psrecord 1034 --log 2000.txt --duration 15 --interval 1
```

Syntax tersebut dijalankan pada terminal kontroler yang sedang dijalankan dan akan menampilkan hasil berupa log dalam folder. PID setiap proses berbeda beda tiap dijalankan, 1034 merupakan PID kontroler ketika dijalankan. Selanjutnya variabel --log 2000.txt digunakan untuk menyimpan hasil log yang direkam dengan nama file 2000.txt, variabel --duration 15 dituliskan untuk merekam log selama 15 detik setelah syntax dijalankan sedangkan variabel --interval 1 merupakan interval yang akan ditampilkan selama durasi yang sudah diberikan, interval 1 menunjukkan jika setiap 1 detik maka log akan ditulis dalam file.

6.1.5.2 Uji CPU dan Memori *Usage* 2 Kontroler POX

Pada pengujian CPU dan Memori *Usage* dengan 2 kontroler ini, psrecord dijalankan dengan memasukkan variabel yang telah ditentukan pada skenario pengujian. Berikut tampilan pengujian yang ditunjukkan pada terminal:

```
$ psrecord 1124 --log 4000.txt --duration 15 --interval 1
```

Untuk pengujian 2 kontroler maka syntax tersebut dijalankan pada terminal kedua kontroler dan *loadbalancing* server yang sedang dijalankan dan akan menampilkan hasil berupa log dalam folder. PID setiap proses berbeda beda tiap dijalankan, 1124 merupakan PID kontroler ketika dijalankan. Selanjutnya variabel --log 4000.txt digunakan untuk menyimpan hasil log yang direkam dengan nama file 4000.txt, variabel --duration 15 dituliskan untuk merekam log selama 15 detik setelah syntax dijalankan sedangkan variabel --interval 1 merupakan interval yang akan ditampilkan selama durasi yang sudah diberikan, interval 1 menunjukkan jika setiap 1 detik maka log akan ditulis dalam file.

6.1.5.3 Uji CPU dan Memori *Usage* 3 Kontroler POX

Pada pengujian CPU dan Memori *Usage* dengan 3 kontroler ini, psrecord dijalankan dengan memasukkan variabel yang telah ditentukan pada skenario pengujian. Berikut tampilan pengujian yang ditunjukkan pada terminal:

```
$ psrecord 1226 --log 8000.txt --duration 15 --interval 1
```

Untuk pengujian 3 kontroler maka syntax tersebut dijalankan pada terminal kedua kontroler dan *loadbalancing* server yang sedang dijalankan dan akan menampilkan hasil berupa log dalam folder. PID setiap proses berbeda beda tiap dijalankan, 1226 merupakan PID kontroler ketika dijalankan. Selanjutnya variabel --log 8000.txt digunakan untuk menyimpan hasil log yang direkam dengan nama file 8000.txt, variabel --duration 15 dituliskan untuk merekam log selama 15 detik setelah syntax dijalankan sedangkan variabel --interval 1 merupakan interval yang akan ditampilkan selama durasi yang sudah diberikan, interval 1 menunjukkan jika setiap 1 detik maka log akan ditulis dalam file.

6.2 Hasil dan Analisis Pengujian

Pada bab ini akan dijelaskan dan dilakukan analisis mengenai hasil pengujian *flow setup rate* dan *flow setup delay* yang telah dilakukan sesuai dengan skenario pengujian yang telah ditentukan. Hasil pengujian akan dianalisis untuk mendapatkan hasil perbandingan dan mengetahui performa terbaik yang dapat dilakukan oleh kontroler POX jika dikombinasikan dengan *loadbalancing*.

6.2.1 Hasil dan Analisis Pengujian Flow Setup Delay dengan Variasi Switch

Pada bab ini akan dijelaskan dan dilakukan analisis mengenai hasil pengujian *Flow Setup Delay* yang telah dilakukan pada bab sebelumnya dimana mengacu pada skenario pengujian, pada akhir sub bab ini akan dilakukan analisis perbandingan yang dihasilkan dari ketiga variasi kontroler yang digunakan dalam pengujian dengan variasi jumlah switch.

6.2.1.1 Hasil dan Analisis Pengujian Latency 1 Kontroler POX

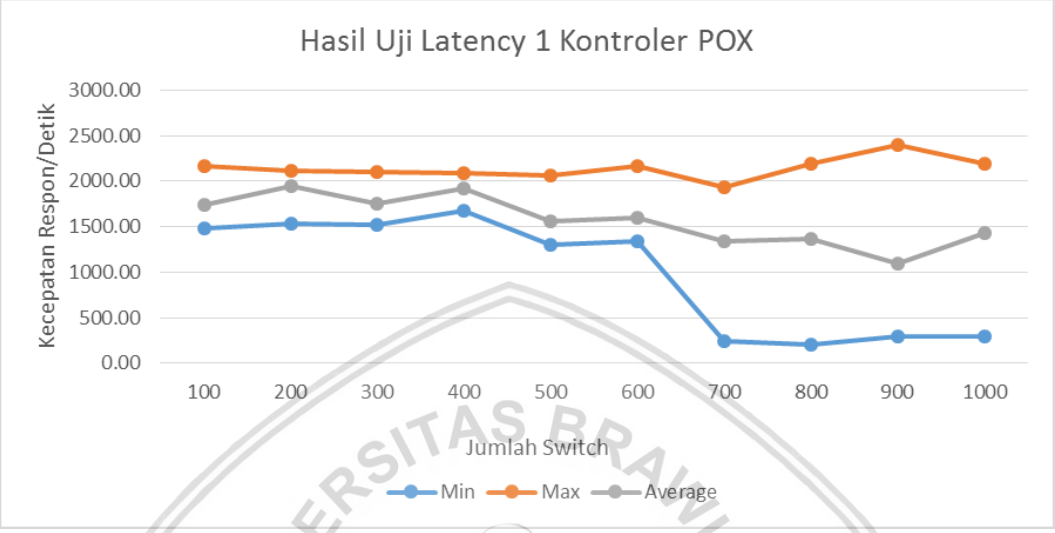
Dari pengujian yang telah dilakukan dengan menggunakan 1 kontroler tanpa menggunakan *loadbalancing* dengan nilai variasi switch dari 100 hingga 1100 pada maksimalnya, maka akan didapatkan hasil seperti pada **Tabel 6.7** berikut:

Tabel 6.7 Hasil Uji Latency 1 Kontroler POX

Hasil Uji Latency 1 Kontroler POX			
Jumlah Switch	Kecepatan Respon (Resp/Detik)		
	Min	Max	Rata-rata
100	1477.99	2163.39	1733.83
200	1528.96	2116.62	1945.06
300	1522.86	2102.75	1750.30
400	1674.95	2090.92	1913.98
500	1294.57	2064.53	1559.44
600	1336.57	2169.53	1600.70
700	238.99	1931.70	1344.32
800	208.73	2190.23	1365.69
900	296.95	2396.76	1098.55
1000	297.78	2186.46	1424.27
1100	null	nulll	null

Dapat dilihat dari pengujian *flow setup delay* yang telah dilakukan menggunakan 1 kontroler POX akan didapatkan data berupa nilai minimal, maksimal, dan rata-rata. Pada tabel pengujian latency 1 kontroler dengan variasi switch diatas didapatkan jumlah switch yang memberikan nilai rata-rata paling tinggi ada pada jumlah 400 switch dengan kecepatan respon sebesar 1913.98 respon/detik.

Nilai *latency* yang dihasilkan dari pengujian menggunakan 1 kontroler berdasarkan tingkat respon yang diberikan kontroler POX setiap detiknya untuk setiap jumlah switch yang dikontrol berada pada kisaran antara 200 hingga 1.600 respon tiap detiknya. Tingkat respon tersebut dapat dilihat dari diagram grafik pada **Gambar 6.21** berikut:



Gambar 6.21 Grafik Hasil Pengujian Latency 1 Kontroler POX

Jika dilihat pada **Gambar 6.21** kecepatan respon dari 1 kontroler POX ini relatif stabil pada angka 200 hingga 1.600 respon per detik, tetapi ketika sudah mencapai jumlah 600 switch dan diberikan penambahan switch hingga mencapai nilai maksimum maka nilai respon akan menurun.

6.2.1.2 Hasil dan Analisis Pengujian Latency 2 Kontroler POX

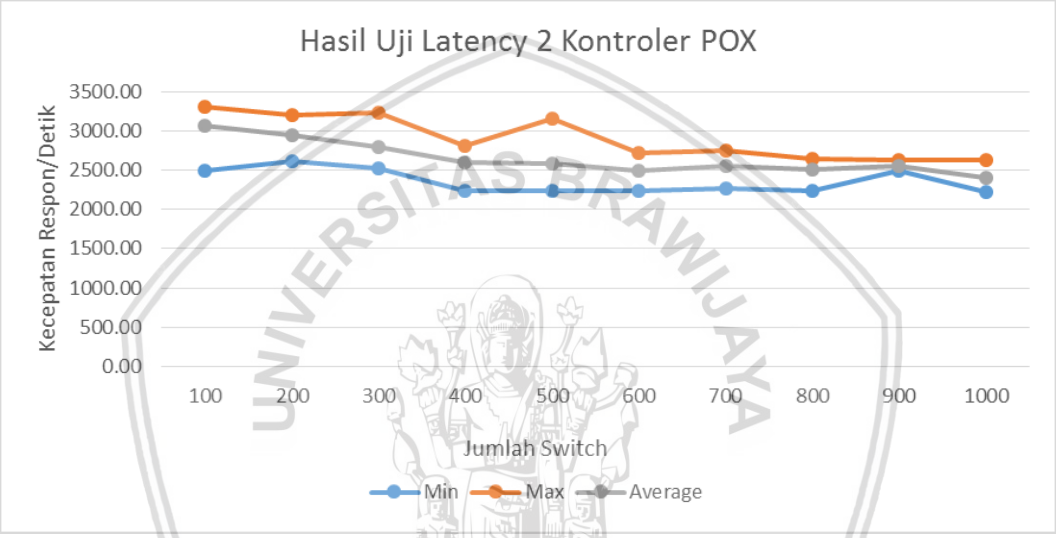
Dari pengujian yang telah dilakukan dengan menggunakan 2 kontroler menggunakan *loadbalancing* dengan nilai variasi switch dari 100 hingga 1100 pada maksimalnya, maka akan didapatkan hasil seperti pada **Tabel 6.8** berikut:

Tabel 6.8 Hasil Uji Latency 2 Kontroler POX

Hasil Uji Latency 2 Kontroler POX			
Jumlah Switch	Kecepatan Respon (Resp/Detik)		
	Min	Max	Rata-rata
100	2497.64	3306.32	3070.33
200	2606.86	3195.08	2944.84
300	2520.91	3236.79	2799.07
400	2232.24	2814.91	2603.75
500	2242.89	3153.95	2579.06
600	2243.59	2712.60	2495.84
700	2266.79	2745.90	2545.27
800	2232.23	2642.90	2501.96
900	2490.48	2632.30	2557.08
1000	2217.02	2622.92	2395.70
1100	null	nulll	null

Dapat dilihat dari pengujian *flow setup delay* yang telah dilakukan menggunakan 2 kontroler POX akan didapatkan data berupa nilai minimal, maksimal, dan rata-rata. Pada tabel pengujian latency 2 kontroler dengan variasi switch diatas didapatkan jumlah switch yang memberikan nilai rata-rata paling tinggi ada pada jumlah 100 switch dengan kecepatan respon sebesar 3070.33 respon/detik.

Nilai *latency* yang dihasilkan dari pengujian menggunakan 2 kontroler berdasarkan tingkat respon yang diberikan kontroler POX setiap detiknya untuk setiap jumlah switch yang dikontrol berada pada kisaran antara 2.200 hingga 3.300 respon tiap detiknya. Tingkat respon tersebut dapat dilihat dari diagram grafik pada **Gambar 6.22** berikut:



Gambar 6.22 Grafik Hasil Pengujian Latency 2 Kontroler POX

Jika dilihat pada **Gambar 6.22** kecepatan respon dari *loadbalancing* kontroler POX 2 kontroler ini relatif stabil pada angka 2.200 hingga 3.300 respon per detik, tetapi ketika sudah mencapai jumlah 100 switch dan diberikan penambahan switch hingga mencapai nilai maksimum maka nilai respon akan menurun.

6.2.1.3 Hasil dan Analisis Pengujian Latency 3 Kontroler POX

Dari pengujian yang telah dilakukan dengan menggunakan 3 kontroler menggunakan *loadbalancing* dengan nilai variasi switch dari 100 hingga 1100 pada maksimalnya, maka akan didapatkan hasil seperti pada **Tabel 6.9** berikut:

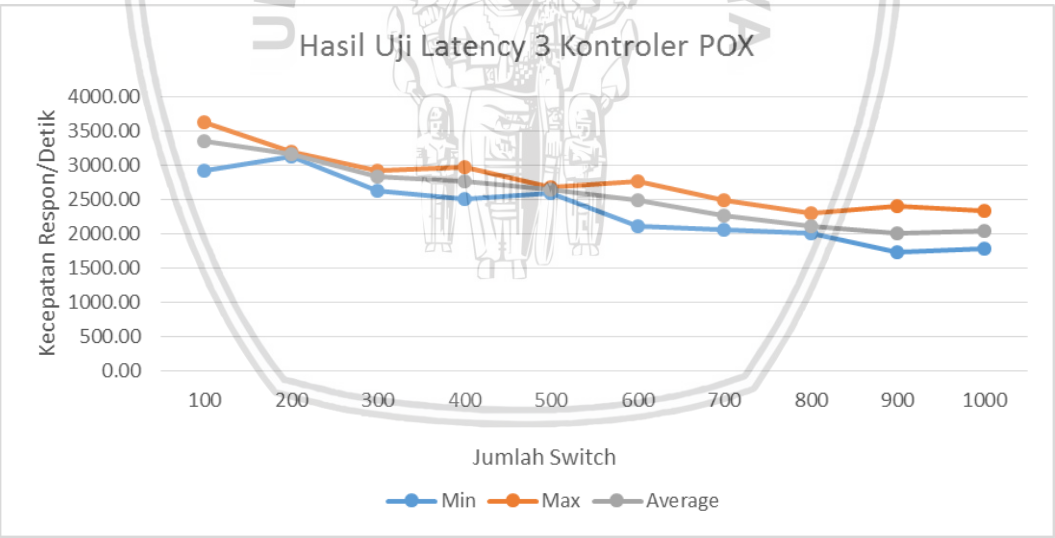
Tabel 6.9 Hasil Uji Latency 3 Kontroler POX

Hasil Uji Latency 3 Kontroler POX			
Jumlah Switch	Kecepatan Respon (Resp/Detik)		
	Min	Max	Rata-rata
100	2915.04	3614.70	3350.03
200	3121.39	3188.92	3149.56
300	2627.85	2914.35	2827.37

400	2500.59	2962.29	2761.06
500	2586.39	2675.89	2636.85
600	2102.20	2753.57	2489.06
700	2064.50	2490.55	2254.97
800	2004.17	2305.69	2101.71
900	1730.46	2393.20	2006.87
1000	1781.40	2330.89	2039.37
1100	null	nulll	null

Dapat dilihat dari pengujian *flow setup delay* yang telah dilakukan menggunakan 3 kontroler POX akan didapatkan data berupa nilai minimal, maksimal, dan rata-rata. Pada tabel pengujian latency 3 kontroler dengan variasi switch diatas didapatkan jumlah switch yang memberikan nilai rata-rata paling tinggi ada pada jumlah 100 switch dengan kecepatan respon sebesar 3350.03 respon/detik.

Nilai *latency* yang dihasilkan dari pengujian menggunakan 3 kontroler berdasarkan tingkat respon yang diberikan kontroler POX setiap detiknya untuk setiap jumlah switch yang dikontrol berada pada kisaran antara 1.700 hingga 3.600 respon tiap detiknya. Tingkat respon tersebut dapat dilihat dari diagram grafik pada **Gambar 6.23** berikut:



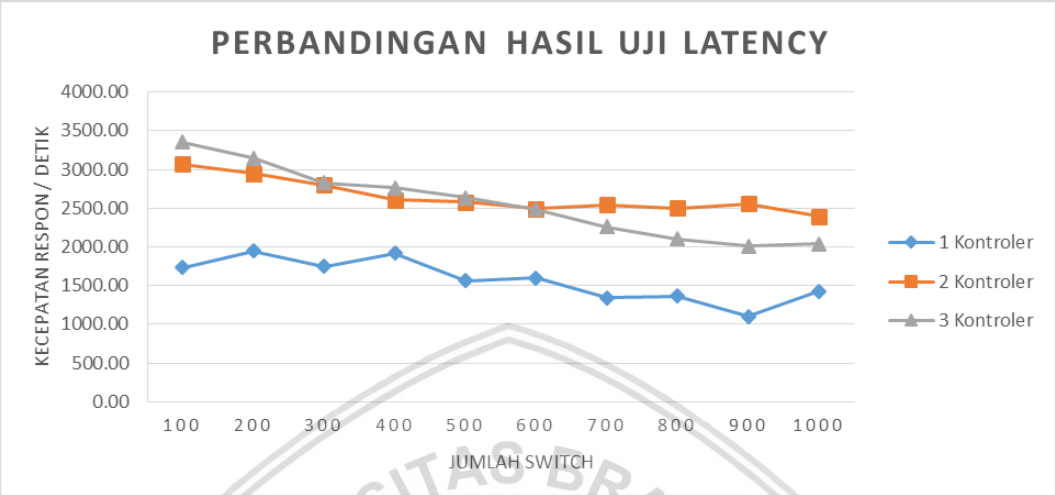
Gambar 6.23 Grafik Hasil Pengujian Latency 3 Kontroler POX

Jika dilihat pada **Gambar 6.23** kecepatan respon dari *loadbalancing* 3 kontroler POX ini relatif stabil pada angka 1.700 hingga 3.600 respon per detik, tetapi ketika sudah mencapai jumlah 100 switch dan diberikan penambahan switch hingga mencapai nilai maksimum maka nilai respon akan menurun.

6.2.1.4 Analisis Perbandingan Hasil Uji Latency dengan Variasi Switch

Hasil uji *latency* dari setiap variasi jumlah *loadbalancing* kontroler POX dan variasi switch memberikan nilai yang berbeda-beda antara satu sama lain. Dari sini

dapat dilihat berapa banyak jumlah kontroler POX yang lebih baik digunakan khuhusnya untuk diberlakukan konsep *loadbalancing* dilihat dari perbandingan hasil uji *latency* dari ketiga varian kontroler tersebut. Perbandingan hasil uji dapat dilihat pada diagram grafik pada **Gambar 6.24** berikut.



Gambar 6.24 Grafik Analisis Perbandingan Hasil Uji Latency

Jika dilihat dari perbandingan yang disajikan pada **Gambar 6.24** diatas maka untuk 1 kontroler cenderung memiliki nilai respon yang tidak terlalu tinggi dibandingkan dengan 2 dan 3 kontroler POX. Nilai respon paling tinggi ada pada 3 kontroler POX dengan kisaran respon antara 1.700 hingga 3.600, tetapi ada penurunan tingkat respon diatas 90 switch. Sama seperti 2 kontroler POX, ketika sudah melewati 100 switch respon yang diberikan cenderung menurun namun untuk jumlah 600 switch hingga 1000 switch, respon yang diberikan 2 kontroler POX ini jauh lebih stabil dan tidak menurun banyak dibandingkan dengan 3 kontroler POX dalam rentang jumlah switch yang sama. Perbedaan nilai respon dari ketiga variasi jumlah kontroler ini disebabkan oleh *loadbalancing* yang digunakan, ketika digunakan pada 2 dan 3 kontroler respon dari kontroler akan meningkat karena setiap kontroler akan mendapatkan pembagian beban kerja yang seimbang.

6.2.2 Hasil dan Analisis Pengujian Flow Setup Delay dengan Variasi Host

Pada bab ini akan dijelaskan dan dilakukan analisis mengenai hasil pengujian *Flow Setup Delay* yang telah dilakukan pada bab sebelumnya dimana mengacu pada skenario pengujian, pada akhir sub bab ini akan dilakukan analisis perbandingan yang dihasilkan dari ketiga variasi kontroler yang digunakan dalam pengujian dengan variasi jumlah host.

6.2.2.1 Hasil dan Analisis Pengujian Latency 1 Kontroler POX

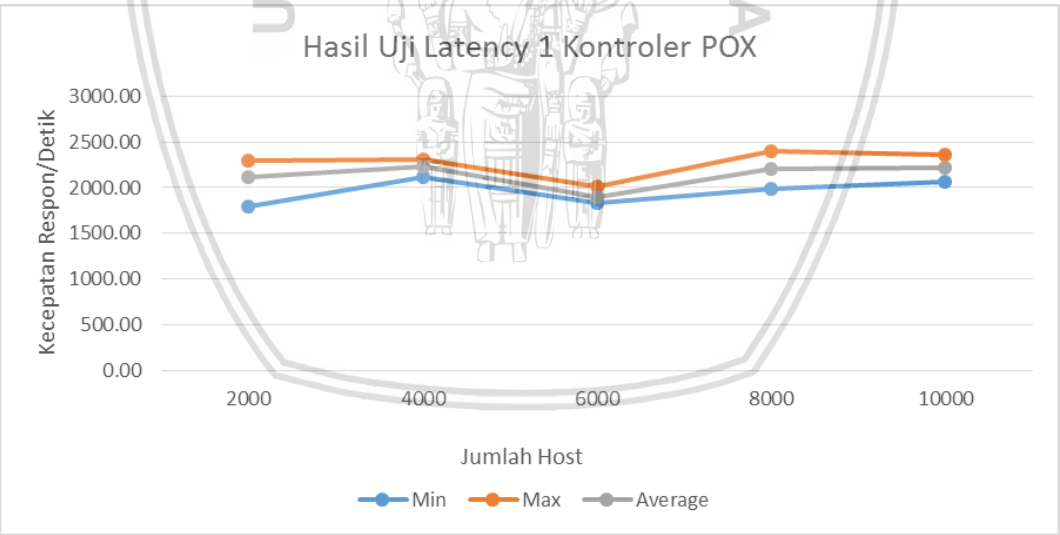
Dari pengujian yang telah dilakukan dengan menggunakan 1 kontroler tanpa menggunakan *loadbalancing* dengan nilai variasi host dari 2000 hingga 10000 pada maksimalnya, maka akan didapatkan hasil seperti pada **Tabel 6.10** berikut:

Tabel 6.10 Hasil Uji Latency 1 Kontroler POX

Hasil Uji Latency 1 Kontroler POX			
Jumlah Host	Kecepatan Respon (Resp/Detik)		
	Min	Max	Rata-rata
2000	1798.00	2291.95	2120.62
4000	2113.99	2306.00	2233.20
6000	1837.79	2013.88	1900.29
8000	1992.76	2395.78	2209.06
10000	2062.94	2356.80	2224.30

Dapat dilihat dari pengujian *flow setup delay* yang telah dilakukan menggunakan 1 kontroler POX akan didapatkan data berupa nilai minimal, maksimal, dan rata-rata. Pada tabel pengujian latency 1 kontroler dengan variasi host diatas didapatkan jumlah host yang memberikan nilai rata-rata paling tinggi ada pada jumlah 4000 host dengan 2233.20 respon/detik.

Nilai *latency* yang dihasilkan dari pengujian menggunakan 1 kontroler berdasarkan tingkat respon yang diberikan kontroler POX setiap detiknya untuk setiap jumlah switch yang dikontrol berada pada kisaran antara 1.700 hingga 2.300 respon tiap detiknya. Tingkat respon tersebut dapat dilihat dari diagram grafik pada **Gambar 6.25** berikut:



Gambar 6.25 Grafik Hasil Pengujian Latency 1 Kontroler POX

Jika dilihat pada **Gambar 6.25** kecepatan respon yang dapat ditangani dari kontroler POX 1 kontroler ini relatif stabil pada angka 1.700 hingga 2.300 respon per detik, mengalami sedikit naik turun tetapi tidak mengalami perubahan yang konstan ketika diberikan penambahan jumlah host.

6.2.2.2 Hasil dan Analisis Pengujian Latency 2 Kontroler POX

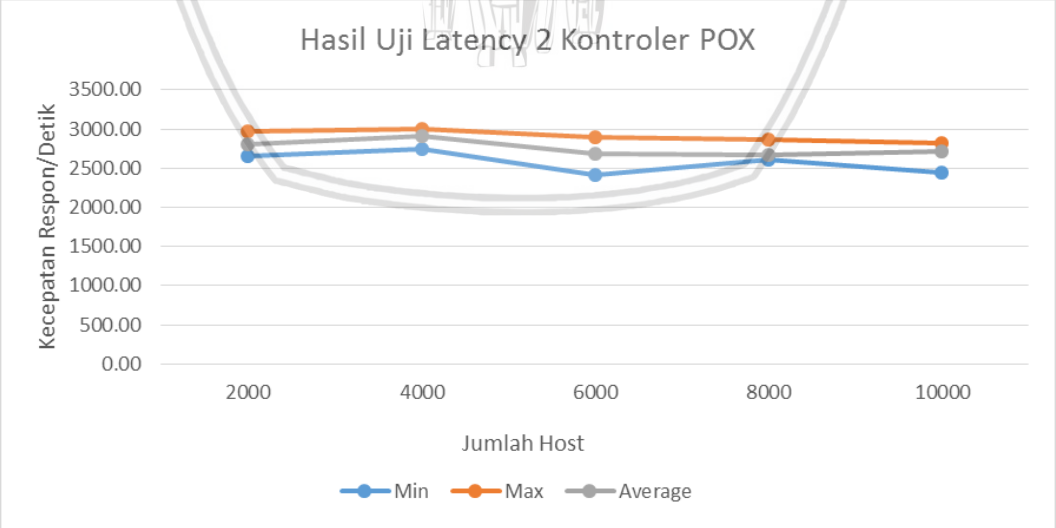
Dari pengujian yang telah dilakukan dengan menggunakan 2 kontroler menggunakan *loadbalancing* dengan nilai variasi host dari 2000 hingga 10000 pada maksimalnya, maka akan didapatkan hasil seperti pada **Tabel 6.11** berikut:

Tabel 6.11 Hasil Uji Latency 2 Kontroler POX

Hasil Uji Latency 2 Kontroler POX			
Jumlah Host	Kecepatan Respon (Resp/Detik)		
	Min	Max	Rata-rata
2000	2654.99	2968.89	2808.47
4000	2751.81	3004.46	2914.90
6000	2419.38	2897.64	2677.03
8000	2603.85	2869.60	2675.84
10000	2438.00	2814.71	2708.43

Dapat dilihat dari pengujian *flow setup delay* yang telah dilakukan menggunakan 2 kontroler POX akan didapatkan data berupa nilai minimal, maksimal, dan rata-rata. Pada tabel pengujian latency 2 kontroler dengan variasi host diatas didapatkan jumlah host yang memberikan nilai rata-rata paling tinggi ada pada jumlah 4000 host dengan 2914.90 respon/detik.

Nilai *latency* yang dihasilkan dari pengujian menggunakan 2 kontroler berdasarkan tingkat respon yang diberikan kontroler POX setiap detiknya untuk setiap jumlah host yang dikontrol berada pada kisaran antara 2.400 hingga 3.000 respon tiap detiknya. Tingkat respon tersebut dapat dilihat dari diagram grafik pada **Gambar 6.26** berikut:



Gambar 6.26 Grafik Hasil Pengujian Latency 2 Kontroler POX

Jika dilihat pada **Gambar 6.26** kecepatan respon yang dapat ditangani dari 2 kontroler POX ini relatif stabil pada angka 2.400 hingga 3.000 respon per detik,

mengalami sedikit naik turun tetapi tidak mengalami perubahan yang konstan ketika diberikan penambahan jumlah host.

6.2.2.3 Hasil dan Analisis Pengujian Latency 3 Kontroler POX

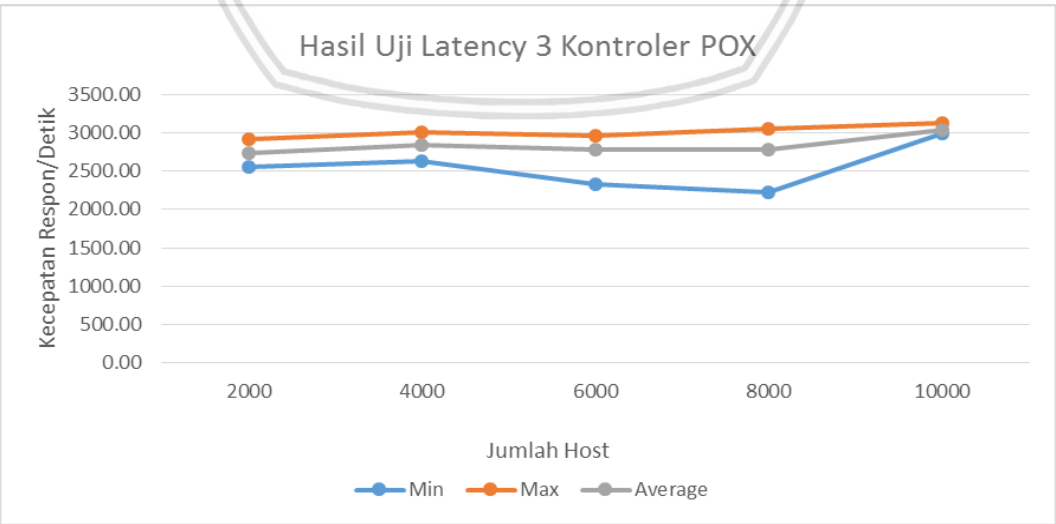
Dari pengujian yang telah dilakukan dengan menggunakan 3 kontroler menggunakan *loadbalancing* dengan nilai variasi host dari 2000 hingga 10000 pada maksimalnya, maka akan didapatkan hasil seperti pada **Tabel 6.12** berikut:

Tabel 6.12 Hasil Uji Latency 3 Kontroler POX

Hasil Uji Latency 3 Kontroler POX			
Jumlah Host	Kecepatan Respon (Resp/Detik)		
	Min	Max	Rata-rata
2000	2550.99	2914.70	2735.36
4000	2637.18	3005.50	2841.91
6000	2336.61	2960.97	2777.56
8000	2225.46	3056.61	2778.40
10000	2993.34	3123.48	3032.85

Dapat dilihat dari pengujian *flow setup delay* yang telah dilakukan menggunakan 3 kontroler POX akan didapatkan data berupa nilai minimal, maksimal, dan rata-rata. Pada tabel pengujian latency 3 kontroler dengan variasi host diatas didapatkan jumlah host yang memberikan nilai rata-rata paling tinggi ada pada jumlah 10000 host dengan 3032.85 respon/detik.

Nilai *latency* yang dihasilkan dari pengujian menggunakan 3 kontroler berdasarkan tingkat respon yang diberikan kontroler POX setiap detiknya untuk setiap jumlah host yang dikontrol berada pada kisaran antara 2.200 hingga 3.000 respon tiap detiknya. Tingkat respon tersebut dapat dilihat dari diagram grafik pada **Gambar 6.27** berikut:

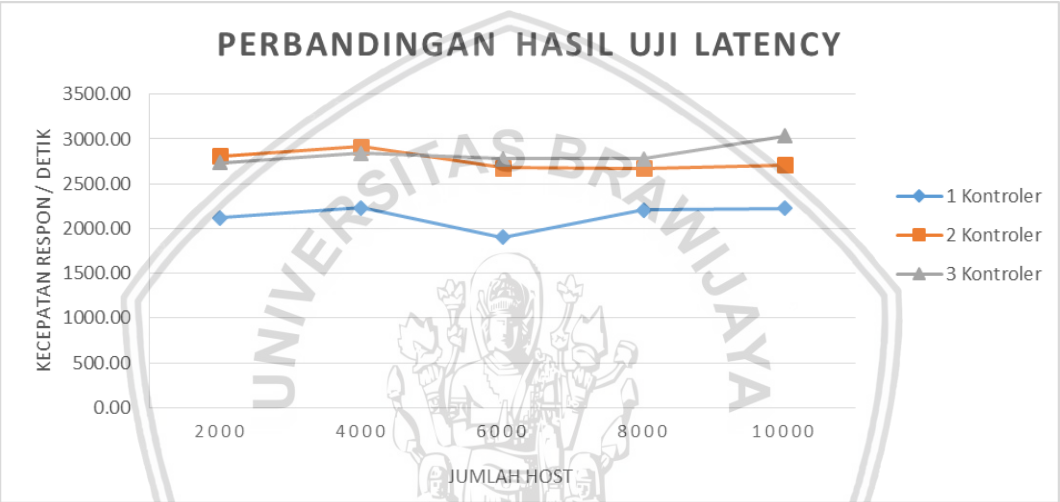


Gambar 6.27 Grafik Hasil Pengujian Latency 3 Kontroler POX

Jika dilihat pada **Gambar 6.27** kecepatan respon yang dapat ditangani dari 3 kontroler POX ini relatif stabil pada angka 2.200 hingga 3.000 respon per detik, mengalami sedikit naik turun tetapi tidak mengalami perubahan yang konstan ketika diberikan penambahan jumlah host.

6.2.2.4 Analisis Perbandingan Hasil Uji Latency dengan Variasi Host

Hasil uji *latency* dari setiap variasi jumlah *loadbalancing* server POX dan variasi host memberikan nilai yang berbeda-beda antara satu sama lain. Dari sini dapat dilihat berapa banyak jumlah kontroler POX yang lebih baik digunakan khuhusnya untuk diberlakukan konsep *loadbalancing* dilihat dari perbandingan hasil uji *latency* dari ketiga varian kontroler tersebut. Perbandingan hasil uji dapat dilihat pada diagram grafik pada **Gambar 6.28** berikut.



Gambar 6.28 Grafik Analisis Perbandingan Hasil Uji Latency

Jika dilihat dari perbandingan yang disajikan pada **Gambar 6.28** diatas maka untuk 1 kontroler cenderung memiliki nilai respon yang tidak terlalu tinggi dibandingkan dengan 2 dan 3 kontroler POX. Nilai respon paling tinggi ada pada 3 kontroler POX dengan kisaran respon rata-rata antara 2.200 hingga 3.000. Sedangkan untuk 2 kontroler POX walaupun respon rata-ratanya ada dibawah 3 kontroler namun respon yang diberikan 2 kontroler POX ini jauh lebih stabil dibandingkan dengan 3 kontroler POX dalam rentang jumlah host yang sama. Perbedaan nilai respon dari ketiga variasi jumlah kontroler ini disebabkan oleh *loadbalancing* yang digunakan, ketika digunakan pada 2 dan 3 kontroler respon dari kontroler akan meningkat karena setiap kontroler akan mendapatkan pembagian beban kerja yang seimbang.

6.2.3 Hasil dan Analisis Pengujian Flow Setup Rate dengan Variasi Host

Pada bab ini akan dijelaskan dan dilakukan analisis mengenai hasil pengujian *Flow Setup Rate* yang telah dilakukan pada bab sebelumnya dimana mengacu pada skenario pengujian, pada akhir sub bab ini akan dilakukan analisis perbandingan yang dihasilkan dari ketiga variasi kontroler yang digunakan dalam pengujian.

6.2.3.1 Hasil dan Analisis Pengujian Throughput 1 Kontroler POX

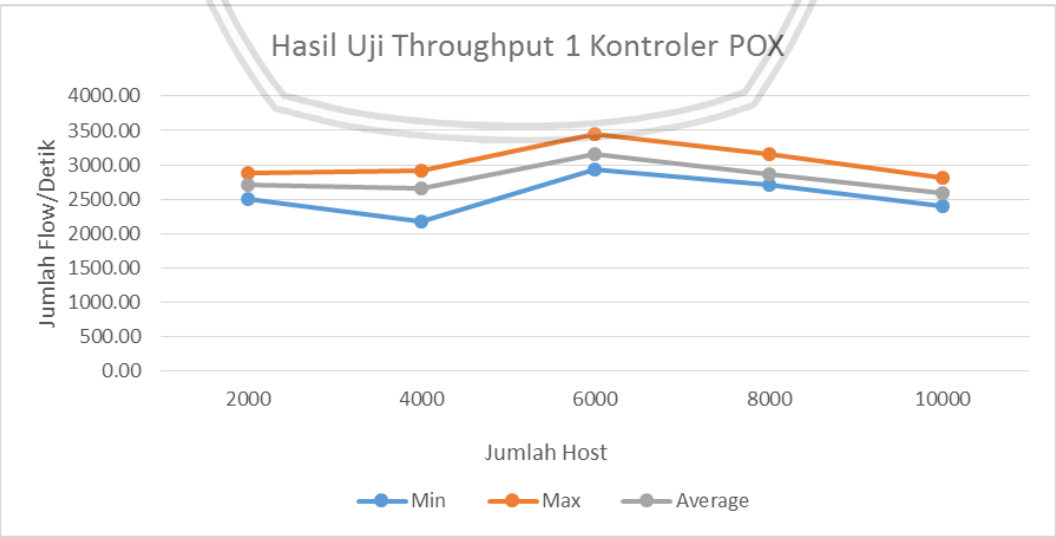
Dari pengujian yang telah dilakukan dengan menggunakan 1 kontroler tanpa menggunakan *loadbalancing* dengan nilai variasi host dari 2000 hingga 10000 pada maksimalnya, maka akan didapatkan hasil seperti pada **Tabel 6.13** berikut:

Tabel 6.13 Hasil Uji Throughput 1 Kontroler POX

Hasil Uji Throughput 1 Kontroler POX			
Jumlah Host	Jumlah aliran data yang dikelola tiap detik (Flow/Detik)		
	Min	Max	Rata-rata
2000	2506.61	2873.68	2704.58
4000	2180.56	2907.40	2656.28
6000	2928.16	3447.54	3156.66
8000	2707.34	3148.38	2863.61
10000	2406.57	2805.12	2594.94

Dapat dilihat dari pengujian *flow setup rate* yang telah dilakukan menggunakan 1 kontroler POX akan didapatkan data berupa nilai minimal, maksimal, dan rata-rata. Pada tabel pengujian throughput 1 kontroler dengan variasi host diatas didapatkan jumlah host yang memberikan nilai rata-rata paling tinggi ada pada jumlah 6000 host dengan jumlah flow yang dapat dikelola sebanyak 3156.66 flow/detik.

Nilai *throughput* yang dihasilkan dari pengujian menggunakan 1 kontroler berdasarkan jumlah flow yang dapat ditangani kontroler POX setiap detiknya untuk setiap jumlah host yang dikontrol berada pada kisaran antara 2.100 hingga 3.400 flow tiap detiknya. Banyak flow yang dapat ditangani tersebut dapat dilihat dari diagram grafik pada **Gambar 6.29** berikut:



Gambar 6.29 Grafik Hasil Pengujian Throughput 1 Kontroler POX

Jika dilihat pada **Gambar 6.29** jumlah flow yang dapat ditangani dari 1 kontroler POX ini relatif stabil pada angka 2.100 hingga 3.400 respon per detik, mengalami sedikit naik turun tetapi tidak mengalami perubahan yang konstan ketika diberikan penambahan jumlah host.

6.2.3.2 Hasil dan Analisis Pengujian Throughput 2 Kontroler POX

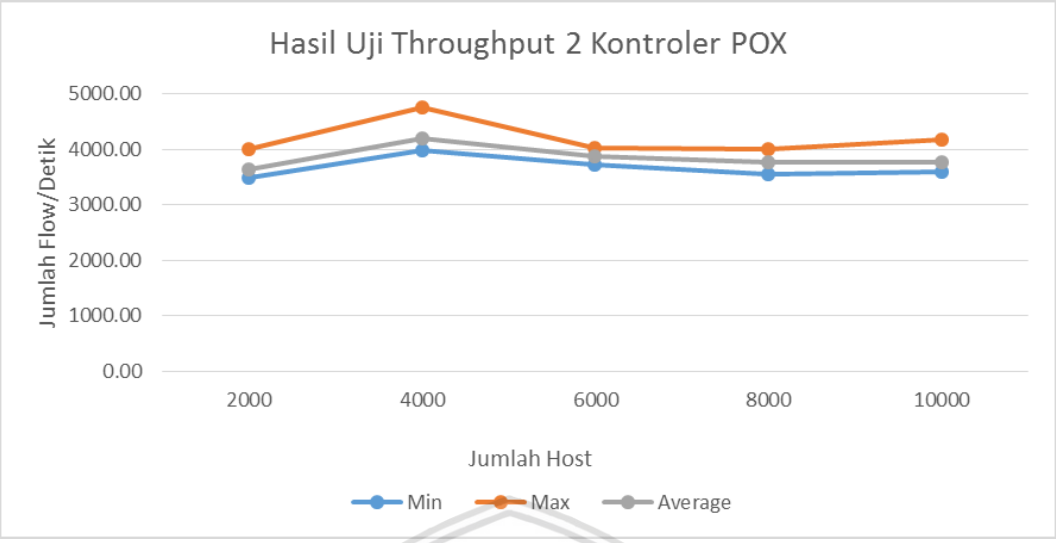
Dari pengujian yang telah dilakukan dengan menggunakan 2 kontroler menggunakan *loadbalancing* dengan nilai variasi host dari 2000 hingga 10000 pada maksimalnya, maka akan didapatkan hasil seperti pada **Tabel 6.14** berikut:

Tabel 6.14 Hasil Uji Throughput 2 Kontroler POX

Hasil Uji Throughput 2 Kontroler POX			
Jumlah Host	Jumlah aliran data yang dikelola tiap detik (Flow/Detik)		
	Min	Max	Rata-rata
2000	3479.85	3996.97	3648.09
4000	3972.63	4745.11	4193.79
6000	3727.34	4032.00	3864.29
8000	3552.42	3992.78	3760.88
10000	3591.68	4165.62	3773.08

Dapat dilihat dari pengujian *flow setup rate* yang telah dilakukan menggunakan 2 kontroler POX akan didapatkan data berupa nilai minimal, maksimal, dan rata-rata. Pada tabel pengujian throughput 2 kontroler dengan variasi host diatas didapatkan jumlah host yang memberikan nilai rata-rata paling tinggi ada pada jumlah 4000 host dengan jumlah flow yang dapat dikelola sebanyak 4193.79 flow/detik.

Nilai *throughput* yang dihasilkan dari pengujian menggunakan 2 kontroler berdasarkan jumlah flow yang dapat ditangani kontroler POX setiap detiknya untuk setiap jumlah host yang dikontrol berada pada kisaran antara 3.400 hingga 4.700 flow tiap detiknya. Banyak flow yang dapat ditangani tersebut dapat dilihat dari diagram grafik pada **Gambar 6.30** berikut:



Gambar 6.30 Grafik Hasil Pengujian Throughput 2 Kontroler POX

Jika dilihat pada **Gambar 6.30** jumlah flow yang dapat ditangani dari *loadbalancing* 2 kontroler POX ini relatif stabil pada angka 3.400 hingga 4.700 respon per detik, mengalami sedikit naik turun tetapi tidak mengalami perubahan yang konstan ketika diberikan penambahan jumlah host.

6.2.3.3 Hasil dan Analisis Pengujian Throughput 3 Kontroler POX

Dari pengujian yang telah dilakukan dengan menggunakan 3 kontroler menggunakan *loadbalancing* dengan nilai variasi host dari 2000 hingga 10000 pada maksimalnya, maka akan didapatkan hasil seperti pada **Tabel 6.15** berikut:

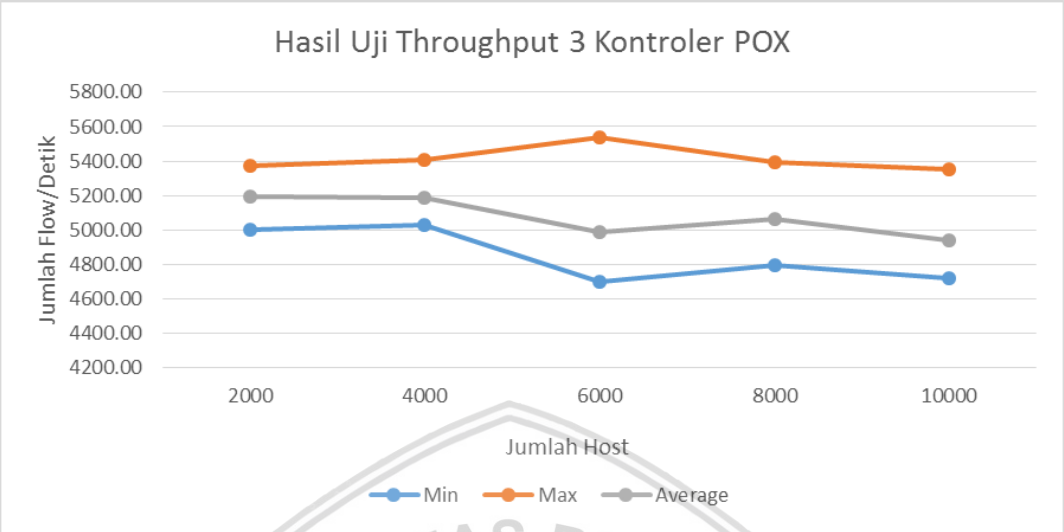
Tabel 6.15 Hasil Uji Throughput 3 Kontroler POX

Hasil Uji Throughput 3 Kontroler POX			
Jumlah Host	Jumlah aliran data yang dikelola tiap detik (Flow/Detik)		
	Min	Max	Rata-rata
2000	5003.03	5372.40	5196.47
4000	5027.67	5403.73	5186.93
6000	4695.36	5539.55	4988.70
8000	4796.15	5393.58	5062.16
10000	4718.69	5350.47	4940.92

Dapat dilihat dari pengujian *flow setup rate* yang telah dilakukan menggunakan 3 kontroler POX akan didapatkan data berupa nilai minimal, maksimal, dan rata-rata. Pada tabel pengujian throughput 3 kontroler dengan variasi host diatas didapatkan jumlah host yang memberikan nilai rata-rata paling tinggi ada pada jumlah 2.000 host dengan jumlah flow yang dapat dikelola sebanyak 5196.47 flow/detik.

Nilai *throughput* yang dihasilkan dari pengujian menggunakan 3 kontroler berdasarkan jumlah flow yang dapat ditangani kontroler POX setiap detiknya untuk setiap jumlah host yang dikontrol berada pada kisaran antara 4.600 hingga

5.500 flow tiap detiknya. Banyak flow yang dapat ditangani tersebut dapat dilihat dari diagram grafik pada **Gambar 6.31** berikut:

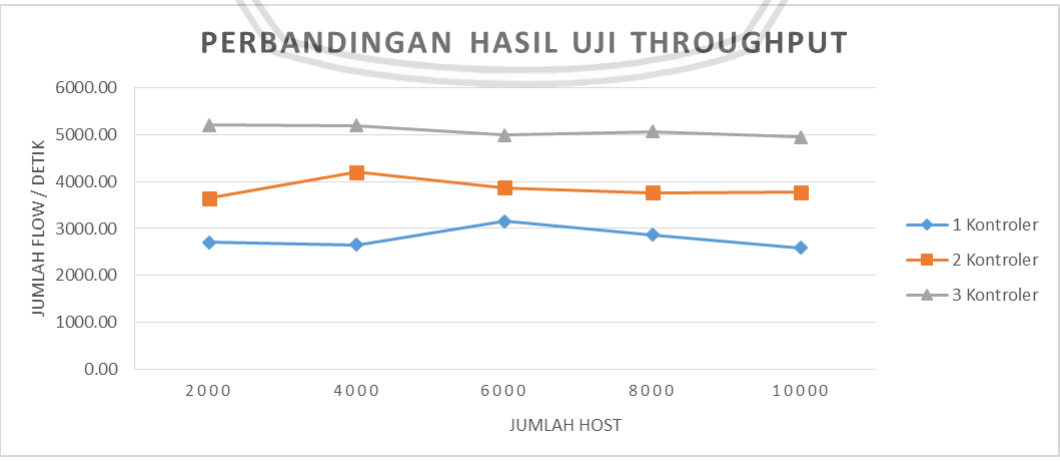


Gambar 6.31 Grafik Hasil Pengujian Throughput 3 Kontroler POX

Jika dilihat pada **Gambar 6.31** jumlah flow yang dapat ditangani dari *loadbalancing* 3 kontroler POX ini relatif stabil pada angka 4.600 hingga 5.500 respon per detik, mengalami sedikit naik turun tetapi tidak mengalami perubahan yang konstan ketika diberikan penambahan jumlah host.

6.2.3.4 Analisis Perbandingan Hasil Uji Throughput dengan Variasi Host

Hasil uji *throughput* dari setiap variasi jumlah *loadbalancing* server POX dan variasi host memberikan nilai yang berbeda-beda antara satu sama lain. Dari sini dapat dilihat berapa banyak jumlah kontroler POX yang lebih baik digunakan khuhusnya untuk diberlakukan konsep *loadbalancing* dilihat dari perbandingan hasil uji *throughput* dari ketiga varian kontroler tersebut. Perbandingan hasil uji dapat dilihat pada diagram grafik pada **Gambar 6.32** berikut.



Gambar 6.32 Grafik Analisis Perbandingan Hasil Uji Throughtput

Jika dilihat dari perbandingan yang disajikan pada **Gambar 6.36** diatas maka untuk 1 kontroler cenderung memiliki jumlah flow yang dapat ditangani tidak terlalu tinggi dibandingkan dengan 2 dan 3 kontroler POX. Jumlah flow paling tinggi yang dapat ditangani ada pada 3 kontroler POX dengan kisaran jumlah flow antara 4.600 hingga 5.500 flow per detik. Dari grafik juga sudah dapat disimpulkan bahwa penambahan kontroler POX akan berpengaruh terhadap jumlah flow yang dapat dikelola tiap detiknya oleh kontroler, semakin banyak kontroler yang digunakan maka akan semakin banyak pula flow yang dapat ditangani. Penambahan jumlah host tidak terlalu berpengaruh dengan performa kontroler, karena cenderung stabil walaupun terkadang nilainya naik turun dan tidak ada nilai penurunan yang konstan. Perbedaan jumlah flow yang dapat ditangani dari ketiga variasi jumlah kontroler ini disebabkan oleh *loadbalancing* yang digunakan, ketika digunakan pada 2 dan 3 kontroler respon dari kontroler akan meningkat karena setiap kontroler akan mendapatkan pembagian beban kerja yang seimbang.

6.2.4 Hasil dan Analisis Pengujian Flow Setup Rate dengan Variasi Switch

Pada bab ini akan dijelaskan dan dilakukan analisis mengenai hasil pengujian *Flow Setup Rate* yang telah dilakukan pada bab sebelumnya dimana mengacu pada skenario pengujian, pada akhir sub bab ini akan dilakukan analisis perbandingan yang dihasilkan dari ketiga variasi server yang digunakan dalam pengujian.

6.2.4.1 Hasil dan Analisis Pengujian Throughput 1 Kontroler POX

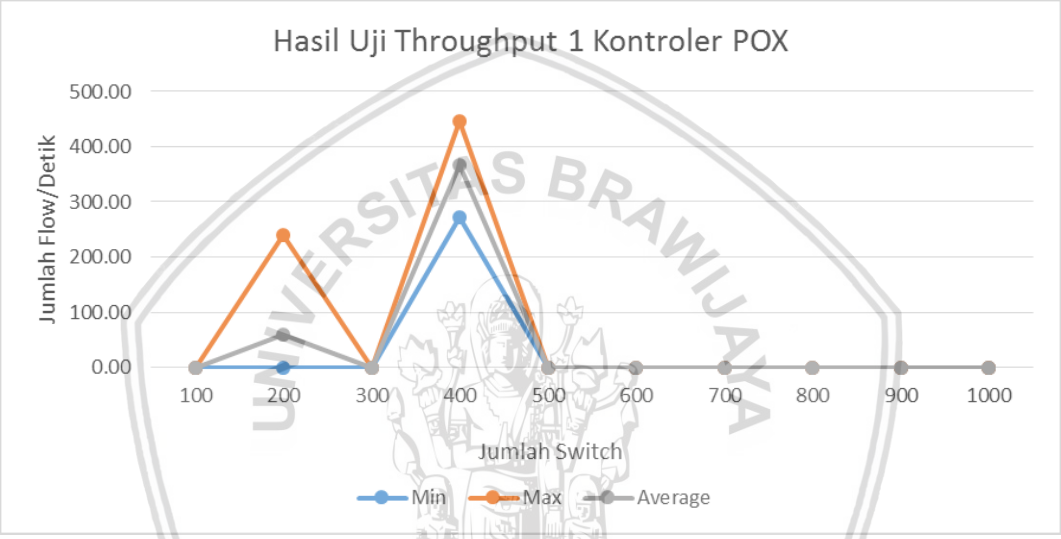
Dari pengujian yang telah dilakukan dengan menggunakan 1 kontroler tanpa menggunakan *loadbalancing* dengan nilai variasi switch dari 100 hingga 1100 pada maksimalnya, maka akan didapatkan hasil seperti pada **Tabel 6.16** berikut:

Tabel 6.16 Hasil Uji Throughput 1 Kontroler POX

Hasil Uji Throughput 1 Kontroler POX			
Jumlah Switch	Jumlah aliran data yang dikelola tiap detik (Flow/Detik)		
	Min	Max	Rata-rata
100	0.00	0.00	0.00
200	0.00	239.00	59.75
300	0.00	0.00	0.00
400	272.94	446.98	367.21
500	0.00	0.00	0.00
600	0.00	0.00	0.00
700	0.00	0.00	0.00
800	0.00	0.00	0.00
900	0.00	0.00	0.00
1000	0.00	0.00	0.00
1100	null	nulll	null

Dapat dilihat dari pengujian *flow setup rate* yang telah dilakukan menggunakan 1 kontroler POX akan didapatkan data berupa nilai minimal, maksimal, dan rata-rata. Pada tabel pengujian throughput 1 kontroler dengan variasi switch diatas didapatkan jumlah switch yang memberikan nilai rata-rata paling tinggi ada pada jumlah 400 switch dengan jumlah flow sebesar 367.21 flow/detik.

Nilai *throughput* yang dihasilkan dari pengujian menggunakan 1 kontroler berdasarkan jumlah flow yang dapat ditangani kontroler POX setiap detiknya untuk setiap jumlah switch yang dikontrol berada pada kisaran antara 200 hingga 400 flow tiap detiknya. Tingkat respon tersebut dapat dilihat dari diagram grafik pada **Gambar 6.33** berikut:



Gambar 6.33 Grafik Hasil Pengujian Throughput 1 Kontroler POX

Jika dilihat pada **Gambar 6.33** jumlah flow dari 1 kontroler POX ini berada pada angka 200 hingga 400 flow per detik, tetapi ketika diberikan penambahan switch hingga mencapai nilai maksimum maka nilai respon akan menurun.

6.2.4.2 Hasil dan Analisis Pengujian Throughput 2 Kontroler POX

Dari pengujian yang telah dilakukan dengan menggunakan 2 kontroler tanpa menggunakan *loadbalancing* dengan nilai variasi switch dari 100 hingga 1100 pada maksimalnya, maka akan didapatkan hasil seperti pada **Tabel 6.17** berikut:

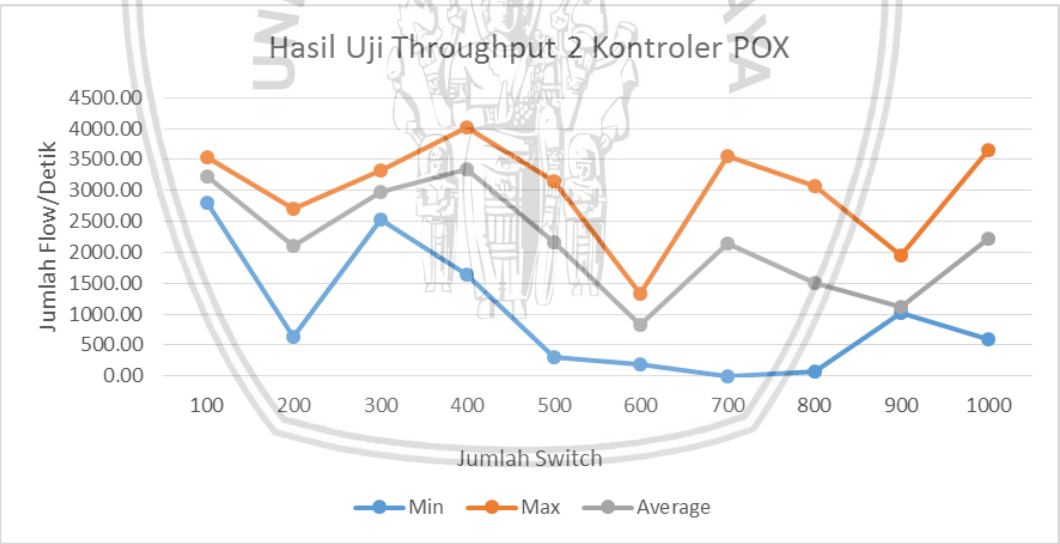
Tabel 6.17 Hasil Uji Throughput 2 Kontroler POX

Hasil Uji Throughput 2 Kontroler POX			
Jumlah Switch	Jumlah aliran data yang dikelola tiap detik (Flow/Detik)		
	Min	Max	Rata-rata
100	2810.50	3540.84	3233.87
200	632.95	2703.23	2109.69
300	2530.77	3315.83	2968.30
400	1640.82	4021.49	3343.30

500	307.43	3150.61	2153.48
600	185.32	1322.96	829.09
700	0.00	3549.01	2142.72
800	70.82	3069.14	1501.10
900	1011.47	1940.00	1112.05
1000	600.93	3659.05	2211.89
1100	null	nulll	null

Dapat dilihat dari pengujian *flow setup rate* yang telah dilakukan menggunakan 2 kontroler POX akan didapatkan data berupa nilai minimal, maksimal, dan rata-rata. Pada tabel pengujian throughput 2 kontroler dengan variasi switch diatas didapatkan jumlah switch yang memberikan nilai rata-rata paling tinggi ada pada jumlah 400 switch dengan jumlah flow sebesar 3343.30 flow/detik.

Nilai *throughput* yang dihasilkan dari pengujian menggunakan 2 kontroler berdasarkan jumlah flow yang dapat ditangani kontroler POX setiap detiknya untuk setiap jumlah switch yang dikontrol berada pada kisaran antara 70 hingga 4.000 flow tiap detiknya. Tingkat respon tersebut dapat dilihat dari diagram grafik pada **Gambar 6.34** berikut:



Gambar 6.34 Grafik Hasil Pengujian Throughput 2 Kontroler POX

Jika dilihat pada **Gambar 6.34** jumlah flow dari 2 kontroler POX ini berkisar pada angka 70 hingga 4.000 flow per detik, tetapi ketika diberikan penambahan switch hingga mencapai nilai maksimum maka nilai respon akan menurun.

6.2.4.3 Hasil dan Analisis Pengujian Throughput 3 Kontroler POX

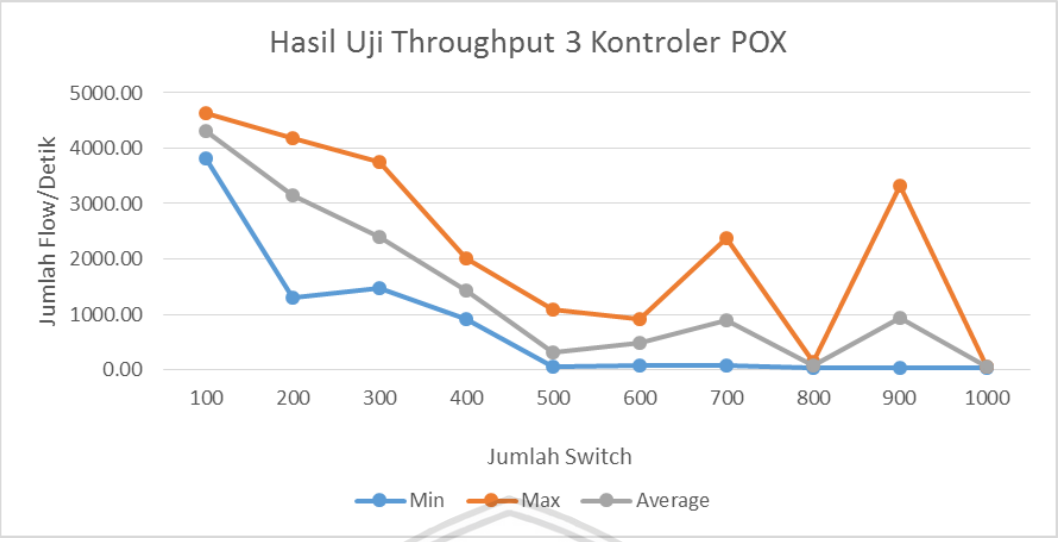
Dari pengujian yang telah dilakukan dengan menggunakan 3 kontroler tanpa menggunakan *loadbalancing* dengan nilai variasi switch dari 100 hingga 1100 pada maksimalnya, maka akan didapatkan hasil seperti pada **Tabel 6.18** berikut:

Tabel 6.18 Hasil Uji Throughput 3 Kontroler POX

Hasil Uji Throughput 3 Kontroler POX			
Jumlah Switch	Jumlah aliran data yang dikelola tiap detik (Flow/Detik)		
	Min	Max	Rata-rata
100	3818.26	4627.45	4314.90
200	1305.40	4180.34	3156.71
300	1480.44	3753.70	2394.87
400	918.31	2009.09	1427.11
500	50.75	1084.10	320.08
600	80.32	919.05	477.78
700	66.30	2372.65	895.57
800	29.18	136.17	70.84
900	30.85	3317.98	925.40
1000	43.07	55.89	48.87
1100	null	nulll	null

Dapat dilihat dari pengujian *flow setup rate* yang telah dilakukan menggunakan 3 kontroler POX akan didapatkan data berupa nilai minimal, maksimal, dan rata-rata. Pada tabel pengujian throughput 3 kontroler dengan variasi switch diatas didapatkan jumlah switch yang memberikan nilai rata-rata paling tinggi ada pada jumlah 100 switch dengan jumlah flow sebesar 4314.90 flow/detik.

Nilai *throughput* yang dihasilkan dari pengujian menggunakan 3 kontroler berdasarkan jumlah flow yang dapat ditangani kontroler POX setiap detiknya untuk setiap jumlah switch yang dikontrol berada pada kisaran antara 29 hingga 4.000 flow tiap detiknya. Tingkat respon tersebut dapat dilihat dari diagram grafik pada **Gambar 6.35** berikut:

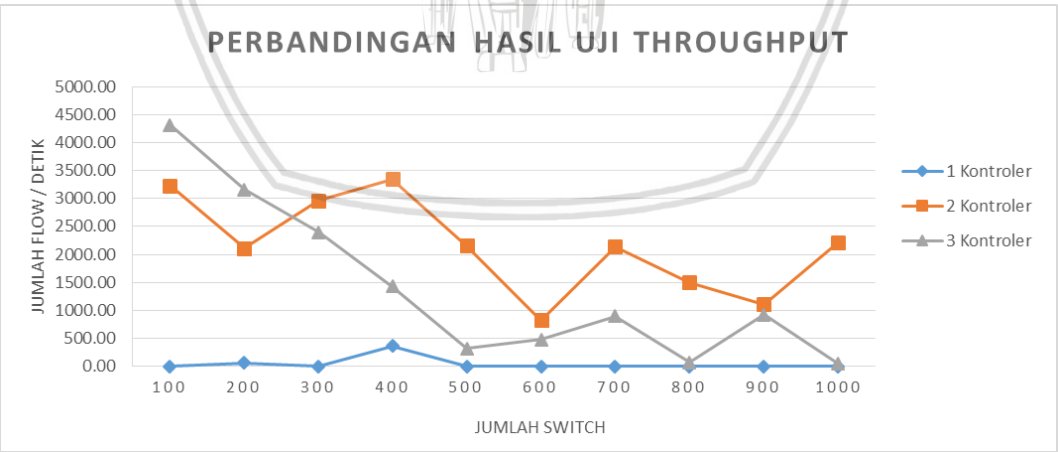


Gambar 6.35 Grafik Hasil Pengujian Throughput 3 Kontroler POX

Jika dilihat pada **Gambar 6.35** jumlah flow dari 3 kontroler POX ini berkisar pada angka 29 hingga 4.000 flow per detik, tetapi ketika diberikan penambahan switch hingga mencapai nilai maksimum maka nilai respon akan menurun.

6.2.4.4 Analisis Perbandingan Hasil Uji Throughput dengan Variasi Switch

Hasil uji *throughput* dari setiap variasi jumlah *loadbalancing* server POX dan variasi host memberikan nilai yang berbeda-beda antara satu sama lain. Dari sini dapat dilihat berapa banyak jumlah kontroler POX yang lebih baik digunakan khuhusnya untuk diberlakukan konsep *loadbalancing* dilihat dari perbandingan hasil uji *throughput* dari ketiga varian server tersebut. Perbandingan hasil uji dapat dilihat pada diagram grafik pada **Gambar 6.36** berikut.



Gambar 6.36 Grafik Analisis Perbandingan Hasil Uji Throughtput

Jika dilihat dari perbandingan yang disajikan pada **Gambar 6.40** diatas maka untuk 1 kontroler cenderung memiliki jumlah flow yang dapat ditangani tidak terlalu tinggi dibandingkan dengan 2 dan 3 kontroler POX. Jumlah flow paling tinggi yang dapat ditangani ada pada 3 kontroler POX dengan kisaran jumlah flow antara 29 hingga 4.000 flow per detik. Dari grafik juga sudah dapat disimpulkan

bahwa penambahan kontroler POX akan berpengaruh terhadap jumlah flow yang dapat dikelola tiap detiknya oleh kontroler, semakin banyak kontroler yang digunakan maka akan semakin banyak pula flow yang dapat ditangani. Penambahan jumlah switch sangat berpengaruh dengan performa kontroler, karena cenderung membuat performa kontroler menjadi menurun jika dilakukan penambahan jumlah switch. Perbedaan jumlah flow yang dapat ditangani dari ketiga variasi jumlah kontroler ini disebabkan oleh *loadbalancing* yang digunakan, ketika digunakan pada 2 dan 3 kontroler respon dari kontroler akan meningkat karena setiap kontroler akan mendapatkan pembagian beban kerja yang seimbang.

6.2.5 Hasil Pengujian CPU dan Memori Usage

Pada bab ini akan dijelaskan dan dilakukan analisis mengenai hasil pengujian CPU dan Memori *Usage* yang telah dilakukan pada bab sebelumnya dimana mengacu pada skenario pengujian dan pada akhir sub bab ini akan dilakukan analisis perbandingan yang dihasilkan dari ketiga variasi kontroler yang digunakan dalam pengujian.

6.2.5.1 Hasil Pengujian CPU dan Memori Usage 1 Kontroler POX

Dari pengujian yang telah dilakukan dengan menggunakan 1 kontroler tanpa menggunakan *loadbalancing* dengan 15 switch dan variasi host, maka akan didapatkan hasil uji CPU dan Memori seperti pada **Tabel 6.19** dan **Tabel 6.20** berikut:

Tabel 6.19 Hasil Uji CPU Usage 1 Kontroler POX

Single Kontroler	
Host	CPU Usage Average (%)
6000	78.3
8000	78.5
10000	78.6

Tabel 6.20 Hasil Uji Memori Usage 1 Kontroler POX

Single Kontroler	
Host	Memory Usage Average (Mb)
6000	24.6
8000	24.6
10000	24.6

Pada hasil uji CPU dan Memori *Usage* 1 kontroler diatas ditunjukkan batas bawah dengan nilai 6000 host batas tengan dengan nilai 80000 host dan batas atas

dengan nilai 500000 host. Rata-rata pada setiap batas akan dibandingkan dengan variasi jumlah kontroler lainnya pada sub bab terakhir.

6.2.5.2 Hasil Pengujian CPU dan Memori Usage 2 Kontroler POX

Dari pengujian yang telah dilakukan dengan menggunakan 2 kontroler menggunakan *loadbalancing* dengan 15 switch dan variasi host, maka akan didapatkan hasil uji CPU dan Memori seperti pada **Tabel 6.21** dan **Tabel 6.22** berikut:

Tabel 6.21 Hasil Uji CPU Usage 2 Kontroler POX

2 Kontroler						
Host	Kontroler 1 (%)	Kontroler 2 (%)	Average (%)	Varian	Std Deviasi	HAproxy / LB (%)
6000	76.9	73.5	75.2	5.848	2.42	12.5
8000	77.2	78.5	77.9	0.845	0.92	13.8
10000	77.5	78.7	78.1	0.720	0.85	14.5

Tabel 6.22 Hasil Uji Memori Usage 2 Kontroler POX

2 Kontroler						
Host	Kontroler 1 (Mb)	Kontroler 2 (Mb)	Average (Mb)	Varian	Std Deviasi	HAproxy / LB (Mb)
6000	21.6	22.2	21.9	0.180	0.42	1.3
8000	21.7	22.2	22.0	0.125	0.35	1.3
10000	21.8	22.0	21.9	0.020	0.14	1.3

Pada hasil uji CPU dan Memori *Usage* 2 kontroler diatas ditunjukkan batas bawah dengan nilai 6000 host batas tengan dengan nilai 80000 host dan batas atas dengan nilai 500000 host. Rata-rata pada setiap batas akan dibandingkan dengan variasi jumlah server lainnya pada sub bab terakhir.

6.2.5.3 Hasil Pengujian CPU dan Memori Usage 3 Kontroler POX

Dari pengujian yang telah dilakukan dengan menggunakan 3 kontroler menggunakan *loadbalancing* dengan 15 switch dan variasi host, maka akan didapatkan hasil uji CPU dan Memori seperti pada **Tabel 6.23** dan **Tabel 6.24** berikut:

Tabel 6.23 Hasil Uji CPU Usage 3 Kontroler POX

3 Kontroler							
Host	Kontroler 1 (%)	Kontroler 2 (%)	Kontroler 3 (%)	Avg (%)	Varian	Std Deviasi	HAproxy / LB (%)
6000	77.1	76.2	66.6	73.3	33.452	5.78	15.0
8000	76.6	76.4	67.9	73.6	24.778	4.98	16.3
10000	75.4	76.5	68.8	73.6	17.343	4.16	16.5

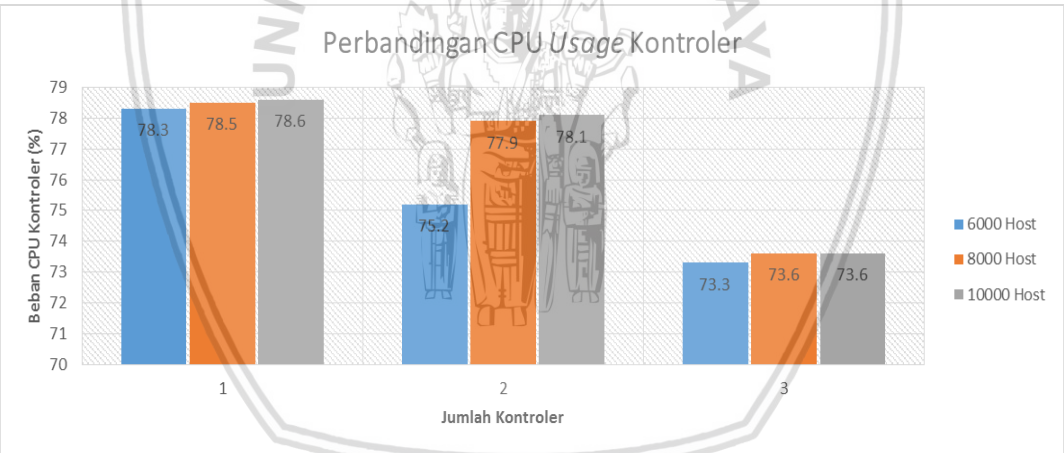
Tabel 6.24 Hasil Uji Memori Usage 3 Kontroler POX

3 Kontroler							
Host	Kontroler 1 (Mb)	Kontroler 2 (Mb)	Kontroler 3 (Mb)	Avg (Mb)	Varian	Std Deviasi	HAproxy / LB (Mb)
6000	19.5	20.1	20.7	20.1	0.337	0.58	1.4
8000	21.0	21.4	21.0	21.1	0.053	0.23	1.4
10000	21.3	21.1	20.9	21.1	0.040	0.20	1.4

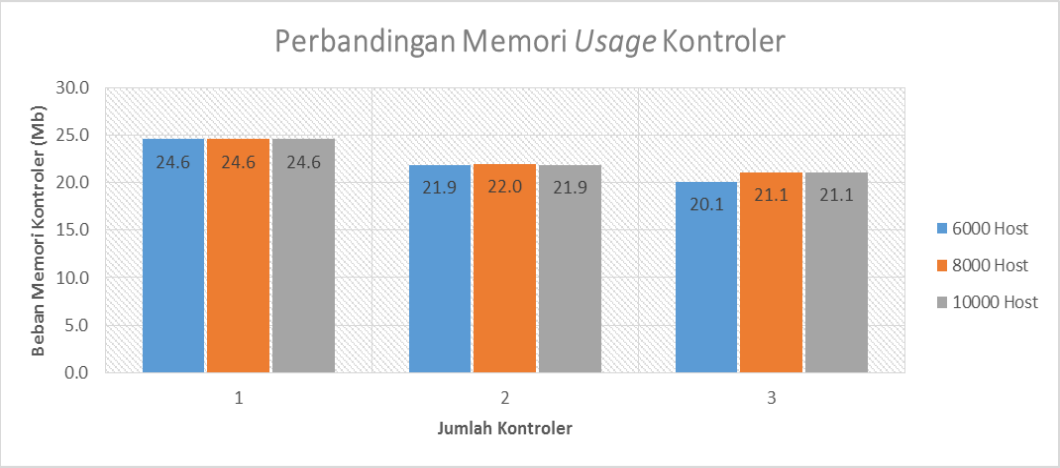
Pada hasil uji CPU dan Memori *Usage* 3 kontroler diatas ditunjukkan batas bawah dengan nilai 6000 host batas tengan dengan nilai 80000 host dan batas atas dengan nilai 500000 host. Rata-rata pada setiap batas akan dibandingkan dengan variasi jumlah kontroler lainnya pada sub bab terakhir.

6.2.5.4 Analisis Perbandingan Hasil Uji CPU dan Memori Usage

Hasil Uji CPU dan Memori Usage dari setiap variasi penggunaan kontroler pada *loadbalancing* kontroler memberikan hasil yang berbeda-beda antara satu sama lain. Dari sini dapat dilihat perbandingan beban dari CPU dan Memori ketika sistem dijalankan. Perbandingan hasil uji CPU dan Memory kontroler *Usage* dapat dilihat pada diagram grafik berikut.



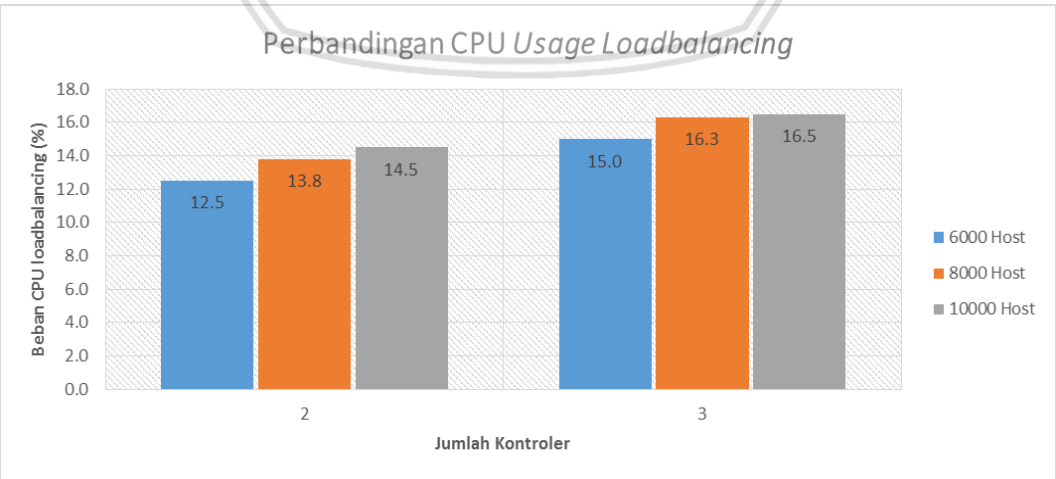
Gambar 6.37 Grafik Perbandingan Hasil Uji CPU Kontroler *Usage*



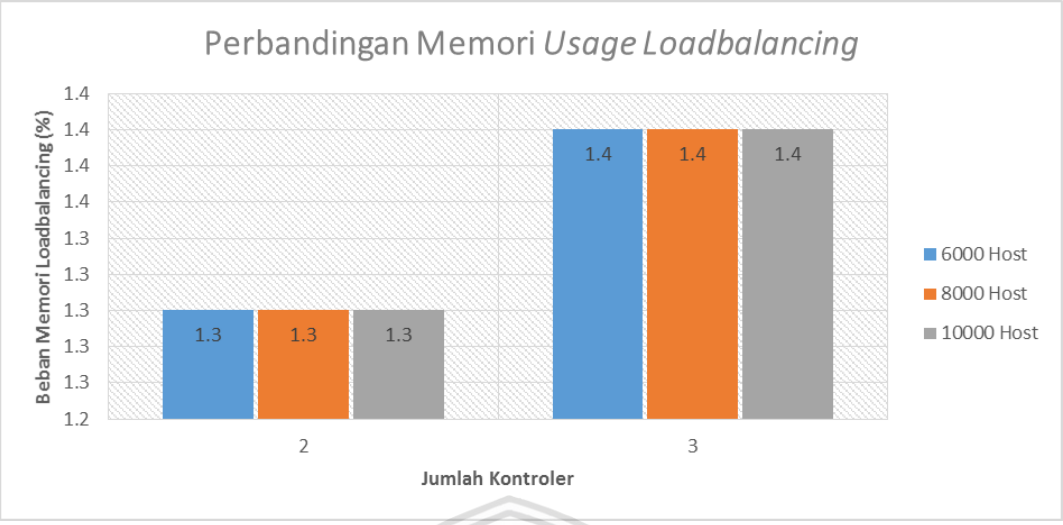
Gambar 6.38 Grafik Perbandingan Hasil Uji Memori Kontroler *Usage*

Jika dilihat pada **Gambar 6.37** dan **Gambar 6.38** maka 3 kontroler jauh lebih unggul dibandingkan penggunaan *single* kontroler dan 2 kontroler. Ini disebabkan karena *loadbalancing* akan membagi beban secara merata, sehingga jika semakin banyak server yang digunakan maka CPU dan memori *usage* dari setiap kontroler akan semakin ringan. Dari perbandingan CPU *usage* pada batas bawah dengan 6000 host, beban *single* kontroler mencapai 78.3% sedangkan pada 3 kontroler beban menurun menjadi 73.3%, begitu pula pada batas atas dengan 10000 host, beban *single* kontroler mencapai 78.6% sedangkan pada 3 kontroler beban menurun menjadi 73.6%. Selanjutnya dilihat dari perbandingan memori *usage* pada batas bawah dengan 6000 host, beban dari *single* kontroler mencapai 24.6 Mb sedangkan pada 3 kontroler beban menurun menjadi 20.1 Mb, begitu pula pada atas atas dengan 10000 host, beban *single* kontroler mencapai 24.6 Mb sedangkan pada 3 kontroler beban menurun menjadi 21.1 Mb.

Dalam pengujian ini juga ditunjukkan perbandingan dari CPU dan Memori usage dari server *loadbalancing,* perbandingan hasil uji dapat dilihat pada **Gambar 6.39** dan **Gambar 6.40** berikut.



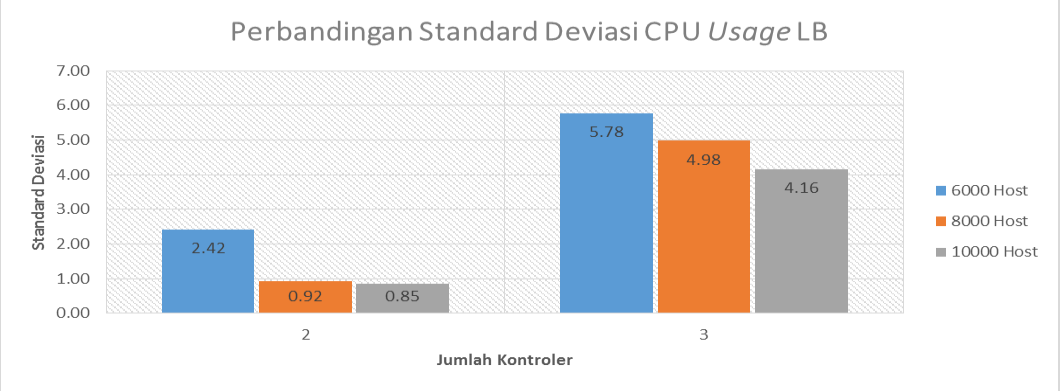
Gambar 6.39 Grafik Perbandingan Hasil Uji CPU *loadbalancing Usage*



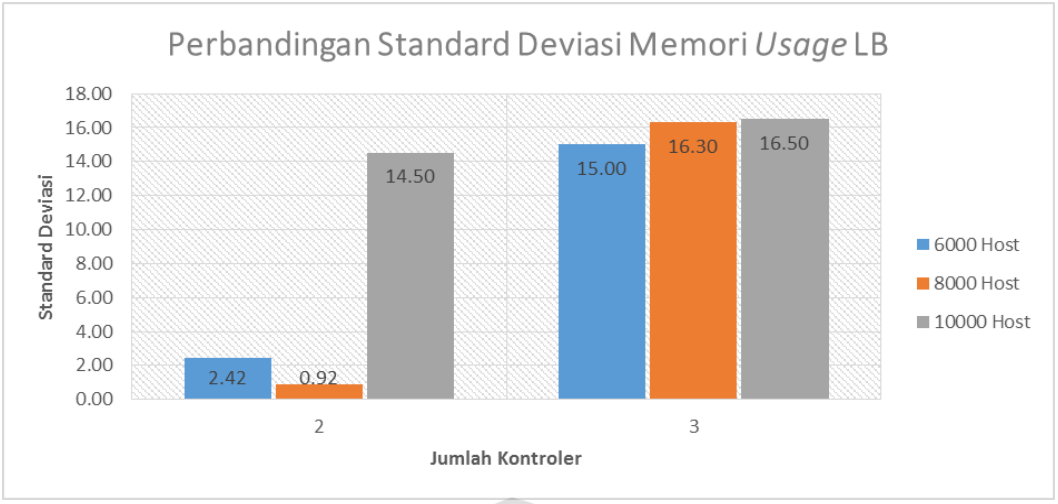
Gambar 6.40 Grafik Perbandingan Hasil Uji Memori *loadbalancing Usage*

Jika dilihat pada **Gambar 6.39** dan **Gambar 6.40** perbandingan CPU *usage* server *loadblancing,* semakin banyak jumlah *loadbalancing* kontroler yang digunakan maka CPU *usage* pada *loadbalancing* server cenderung meningkat dimana CPU *usage* pada batas bawah dengan 6000 host untuk 2 server adalah 12.5% sedangkan untuk 3 server beban meningkat menjadi 15.0%. Jika dilihat pada batas atas dengan 10000 host untuk 2 server adalah 14.5% sedangkan untuk 3 server beban meningkat menjadi 16.5%, hal ini disebabkan karena kontroler yang ditangani oleh server *loadbalancing* semakin banyak sehingga beban dari CPU akan meningkat seiring dengan bertambahnya server yang ditangani. Begitu juga dengan memori *usage* dimana dengan jumlah kontroler sebanyak 2, beban dari server *loadbalancing* berkisar pada 1.3Mb sedangkan dengan jumlah 3 kontroler yang ditangani beban meningkat menjadi 1.4Mb pada semua batas.

Pada CPU dan memori *usage* dapat kita ambil informasi berupa standard deviasi, dimana standard deviasi ini merupakan sebaran nilai yang dapat dilihat pada penggunaan 2 server dan 3 server. Semakin kecil hasilnya maka semakin merata juga pembagian beban pada setiap kontroler. Perbandingan standard deviasi tersebut dapat dilihat pada **Gambar 6.41** dan **Gambar 6.42** berikut.



Gambar 6.41 Grafik Perbandingan Standar Deviasi CPU *Usage*



Gambar 6.42 Grafik Perbandingan Standar Deviasi Memori *Usage*

Jika dilihat dari **Gambar 6.41** dan **Gambar 6.42** untuk pembagian beban CPU *usage* dari penggunaan 2 server lebih unggul dibanding 3 server, yang artinya setiap server memiliki beban yang hampir sama. Begitu juga pada memori *usage* persebarannya juga lebih unggul 2 server dibandingkan dengan 3 server. Perbedaan persebaran ini diakibatkan oleh algoritma yang digunakan sebagai *loadbalancing,* karena pada penelitian ini digunakan algoritma *round robin* dimana pembagiannya akan diurutkan dengan cara bergilir sehingga untuk jumlah server ganjil pembagiannya pasti akan ada yang berat di satu titik. Sehingga untuk jumlah server genap yaitu 2 server pada penelitian ini, persebaran untuk pemerataan beban pada setiap servernya akan lebih unggul pada 2 server.

BAB 7 KESIMPULAN DAN SARAN

7.1 Kesimpulan

Berdasarkan hasil perancangan, implementasi, pengujian dan analisis dari implementasi *load balancing* kontroler *Software Defined Network* dapat disimpulkan bahwa:

1. Penerapan *load balancing* pada kontroler Software Defined Network digunakan untuk membagi beban antara kontroler yang menangani request dari client yang terhubung dalam jaringan. Pembagian beban berdampak pada performa kontroler POX dalam menangani request. Terbukti dari pengujian CPU dan Memori *Usage*, dimana seiring bertambahnya jumlah server kontroler maka kinerja tiap kontroler akan semakin ringan. Tetapi seiring bertambahnya kontroler maka beban dari server *load balancing* justru meningkat.
2. Berdasarkan serangkaian pengujian dan analisis yang telah dilakukan, *load balancing* pada kontroler POX akan memberikan performa terbaik dalam waktu respon ketika menggunakan 3 kontroler. Dari pengujian dengan variasi switch didapatkan kisaran respon yang antara 1.700 hingga 3.600 repon per detik ini jauh lebih baik dibandingkan hasil dari menggunakan 2 kontroler dan 1 kontroler. Dari pengujian dengan variasi host ketiga variasi kontroler cenderung stabil, tetapi tetap lebih unggul 3 kontroler karena memberikan respon tertinggi pada kisaran antara 2.200 hingga 3.000 respon per detik. Peningkatan respon ini diakibatkan dari penggunaan *loadbalancing* pada 2 dan 3 kontroler, sehingga performa yang didapatkan semakin meningkat seiring bertambahnya kontroler yang digunakan.
3. Berdasarkan serangkaian pengujian dan analisis yang telah dilakukan, *load balancing* pada kontroler POX akan memberikan performa terbaik dalam menangani flow ketika menggunakan 3 kontroler. Dari pengujian dengan variasi host didapatkan kisaran jumlah flow yang dapat ditangani kontroler antara 4.600 hingga 5.500 flow per detik ini jauh lebih baik dibandingkan hasil dari menggunakan 2 kontroler dan 1 kontroler. Jumlah flow yang ditangani dari ketiga varian *load balancing* kontroler ini cenderung stabil, dari jumlah host yang paling kecil sebanyak 2000 host hingga yang terbanyak 10.000 host tidak mengalami peningkatan dan penurunan yang terlalu drastis. Dari pengujian dengan variasi switch hasil yang didapatkan tetap menunjukkan 3 kontroler lebih baik, tetapi nilai jumlah flow yang dapat ditangani tidak terlalu bagus seiring dengan bertambahnya jumlah switch untuk ketiga variasi jumlah kontroler. Peningkatan jumlah flow yang dapat ditangani ini diakibatkan dari penggunaan *loadbalancing* pada 2 dan 3 kontroler, sehingga performa yang didapatkan semakin meningkat seiring bertambahnya kontroler yang digunakan.

7.2 Saran

Setelah menyelesaikan penelitian ada beberapa saran yang dapat disampaikan oleh penulis untuk penelitian lebih lanjut terkait dengan penelitian ini adalah:

1. Perlu dilakukan pengujian dengan menambahkan jumlah kontroler diatas 3 mesin agar dapat dibandingkan dengan hasil yang telah didapatkan dengan penelitian ini.
2. Perlu dilakukan penelitian sejenis dengan menggunakan kontroler yang berbeda dan yang stabil dengan Simulator Cbench seperti pada penelitian ini agar bisa menjadi refrensi lebih lanjut sebagai pemilihan kontroler yang cocok digunakan untuk diterapkan *load balancing* dengan berbagai kondisi dan kompleksitas jaringannya.

DAFTAR PUSTAKA

Al-Shabibi, A., 2013. FlowVisor. [online] Tersedia di : https://github.com/OPENNETWORKINGLAB/flowvisor/wiki? [Diakses 6 Maret 2017].

Al-Shabibi, A., dan McCauley, M., 2015. POX Wiki. [online] Tersedia di : https://openflow.stanford.edu/display/ONL/POX+Wiki#POXWiki-Requirements [Diakses 5 Oktober 2017].

Bailey, S., Bansal, D., dan Dunbar, L., 2013. SDN Architecture Overview. Open Networking Foundation.

Chao, H.J., dan Bin, L., 2007. High Performance Switches And Routers. Willey Interscience : A John Willey & Sons Inc.

Erickson, D. 2013. The Beacon OpenFlow Controller. Stanford University : ACM.

Goransson, P. dan Black, C., 2014. Software Defined Network : A Comprehensive Approach. Waltham, USA : Elsevier Inc.

Guo, Z., Su, M., dan Duan, Z., 2014. Improving The Performance of Load Balancing in Software Defined Network Through Load Variance Based Synchronization. USA : Elsevier Inc.

Hu, F., Qi Hao., dan Ke Bao., 2014. A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation. IEEE Comuunication Surveys and Tutorial: IEEE Xplore Digital Library.

Jose, L., Garcia, G., dan Aguilera, G., 2014. A New Probabilistic Extension of Dijkstra's Algorithm to Simulate More Realistic Trafic Flow in a Smart City. University of Milaga, Spain : Elsevier Inc.

Muntaner, Guillermo R.d.T. 2012. Evaluation of OpenFlow Controllers. [online] Tersedia di : <http://www.valleytalk.org/wp-content/uploads/2013/02/Evaluation_Of_OF_Controllers.pdf> [Diakses 7 Maret 2017].

Sherwood, R., dan Gibb, G., 2009. FlowVisor : A Network Virtualize Layer. Stanford University, USA : Deutsche Telekom Inc. R & D Lab.

Sonba, A., dan Hassan, A., 2014. Performance Comparisson Of The State of The Art OpenFlow Controllers. Master Thesis in Computer Network Engineering: Computer and Electrical Engineering Halmstad University.

HAproxy. 2001. HAproxy The Reliable, High Performance TCP/HTTP Load Balancer. [online] Tersedia di: <http://www.haproxy.org/> [Diakses 10 Oktober 2017].

Vengainathan, B., Anton, B., dan Vishwas, M., 2014. Benchmarking Methodology for OpenFlow SDN. [online] Tersedia di : <https://tools.ietf.org/html/draft-bhuvan-bmwg-of-controller-benchmarking-00> [Diakses 6 Maret 2017].

Vinayagamurthy, D., Balasundaram, J., 2012. Load Balancing between Controllers. University of Toronto, Canada : Department of Computer Science.