# IMPLEMENTASI SISTEM *LOAD BALANCING* MENGGUNAKAN METODE ROUND ROBIN DENGAN *CONTROLLER* OPENDAYLIGHT SEBAGAI KOMPONEN UTAMA ARSITEKTUR SDN

## *Implementation Load Balancing system using Round Robin method with Opendaylight controller as main component of SDN*

**Edgar Giovanni Ariestawan[1], Nachwan Mufti Adriansyah[2], Ridha Muldina Negara[2]**

[1,2,] Prodi S1 Teknik Telekomunikasi, Fakultas Teknik Elektro, Universitas Telkom,

[1]edgargiovanni@students.telkomuniversity.ac.id, [2]nachwanmufti@telkomuniversity.ac.id, [2]ridhanegara@telkomuniversity.ac.id

**ABSTRAK**

***Software Defined Network* (SDN) adalah suatu paradigma yang merubah cara merancang, mengatur, dan mengontrol jaringan. SDN membuat suatu jaringan dapat diprogram sesuai dengan kebutuhan yang ada. Salah satu protokol yang mendukung SDN yaitu *OpenFlow*. Pada protokol *OpenFlow*, terdapat perangkat kontrol (*control plane*) dan perangkat penyalur paket data (*data plane*). *Control Plane* berfungsi sebagai pengontrol jaringan yang akan dilewatkan, *Data Plane* berfungsi untuk mengirimkan paket data ke tujuannya. Dalam tugas akhir ini, akan dirancang sebuah sistem yang memiliki fungsi *load balancing* dengan menggunakan metode *Round Robin* dan OPENDAYLIGHT sebagai SDN *controller*. Dalam pengimplementasiannya akan dilakukan pengujian pada sebuah jaringan komputer dengan jumlah *server* sebanyak 3 unit dan jumlah *client* sebanyak 16 *unit*. Masing-masing *client* melakukan koneksi secara acak dengan cara mengirimkan paket *Internet Control Message Protocol* (ICMP) terhadap alamat IP publik dan *controller* OPENDAYLIGHT melakukkan pemetaan *client* tersebut terhadap *server* berdasarkan metode *round robin*. Kemudian dilakukan uji coba dengan beberapa parameter seperti *response time, network convergence, overhead,* dan *resource utilization* yang digunakan dan melakukan perbandingan bagaimana perbedaan antara sebelum menggunakan *load balancing* dan sesudah menggunakan *load balancing.***

**Kata kunci: *sofware defined network, openflow, Round Robin*, OPENDAYLIGHT, *load balancing*.**

**ABSTRACT**

***Software Defined Network (SDN) is a paradigm that changes how to design, manage, and control the network. SDN makes a network can be programmed in accordance with existing needs. One of the protocols that supports SDN is OpenFlow. In the OpenFlow protocol, there is a control plane and a data plane. Control Plane serves as a network controller to be missed, Data Plane serves to send data packets to its destination. In this final project, we will design a system that has load balancing function using Round Robin and OPENDAYLIGHT method as SDN controller. In the implementation will be tested on a computer network with the number of servers as much as 3 units and the number of clients as many as 15 units. Each client makes a random connection by sending an Internet Control Message Protocol (ICMP) packet to the public IP address and the OPENDAYLIGHT controller to mapping the client against the server based on the round robin method. Then tested with some parameters such as response time, network convergence, overhead, and resource utilization, then we comparison how difference between before using load balancing and after using load balancing***

***Keywords: software defined network, openflow, Round Robin, OPENDAYLIGHT, load balancing.***

## 1. Pendahuluan

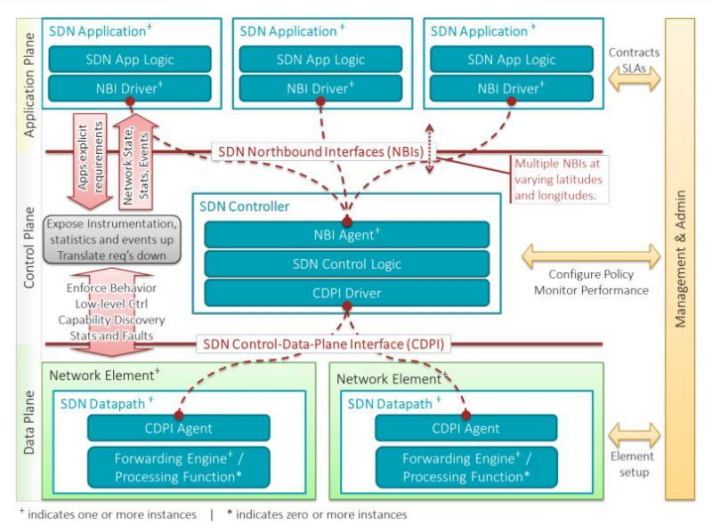
Pemisahan *data plane* dan *control plane* pada perangkat jaringan komputer seperti *router* dan *switch* memungkinkan untuk memprogram perangkat tersebut sesuai dengan yang diiinginkan secara terpusat. Pemisahan inilah yang mendasari terbentuknya paradigma baru dalam jaringan komputer yang disebut *Software Defined Networking* (SDN). Dengan cara ini pengguna *Software Defined Networking* (SDN) dapat membangun sendiri sebuah aplikasi maupun gabungan dari beberapa aplikasi yang akan dijalankan pada *platform controller*. *Platform controller* menyediakan *Application Programming Interfaces* (APIs) sehingga memudahkan dalam mengimplementasikan fitur dan layanan dalam jaringan komputer. Pada *Software Defined Networking* (SDN), *controller* terpusat mengkonfigurasi tabel *forwarding (flow-table) Switch* yang bertanggung jawab untuk meneruskan aliran paket komunikasi. Salah satu contoh fitur yang biasanya telah tersedia pada sebuah perangkat *switch* ataupun *router* adalah *load balancing* dengan metode yang sudah diterapkan oleh *vendor* penyedianya.

*Load balancing* merupakan suatu teknik untuk membagi beban trafik yang diterima kepada beberapa *server* dengan tujuan agar *server* tidak mengalami *overload* dan tetap menyediakan layanan dengan baik. Teknik *load balancing* dapat diterapkan dengan menggunakan beberapa metode antrian, antara lain: *round robin, weighted round robin, persistent, hash based*. Sebagai salah satu contoh adalah penerapan metode *round robin* pada teknik *load balancing* dimana metode ini membagi beban trafik secara bergiliran dan berurutan dari satu *server* ke *server* lain. Kehebatan teknologi *Software Defined Networking* (SDN) dalam sistem jaringan komputer ini dianggap menarik oleh penulis, penulis sangat tertarik untuk mengimplementasikan teknik *Load Balancing* dengan metode *round robin* pada *Software Defined Networking* (SDN) menggunakan *controller* OPENDAYLIGHT. [1][2]

## 2. Dasar Teori

### 2.1..Arsitektur dan Karakteristik SDN

Dalam konsep SDN, tersedia *open interface* yg memungkinkan sebuah entitas *software*/aplikasi untuk mengendalikan konektivitas yang disediakan oleh sejumlah sumber-daya jaringan, mengendalikan aliran trafik yang melewatinya serta melakukan inspeksi terhadap atau memodifikasi trafik tersebut. Gambar berikut menunjukan arsitektur SDN beserta komponen dan interaksinya. [5]




*Gambar 2.1.Arsitektur SDN*

Arsitektur SDN dapat dilihat sebagai 3 lapis/bidang: [5]

- **infrastruktur** (*data-plane* / *infrastructure layer*): terdiri dari elemen jaringan yg dapat mengatur *SDN Datapath* sesuai dengan instruksi yg diberikan melalui *Control-Data-Plane Interface* (CDPI).

- **kontrol** (*control plane / layer*): entitas kontrol (*SDN Controller*) mentranslasikan kebutuhan aplikasi dengan infrastruktur dengan memberikan instruksi yg sesuai untuk *SDN Datapath* serta memberikan informasi yg relevan dan dibutuhkan oleh *SDN Application.*
- **aplikasi** (*application plane / layer*): berada pada lapis teratas, berkomunikasi dengan sistem via *NorthBound Interface* (NBI).

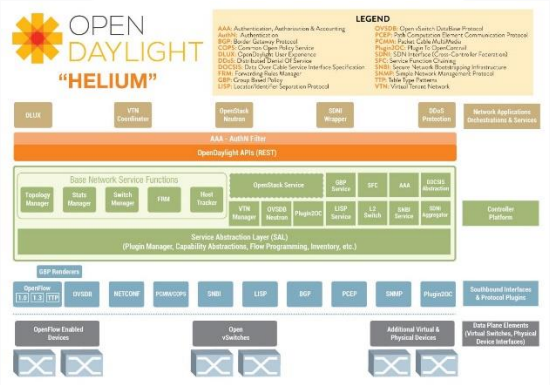
Bidang ***Management & Admin*** bertanggung-jawab dalam: inisiasi elemen jaringan, memasangkan *SDN Datapath* dengan *SDN Controller*, atau menkonfigurasi cakupan (*coverage*) dari *SDN Controller* dan *SDN App.*

## 2.2.OPENDAYLIGHT Controller

*OpenDaylight Controller* adalah sebuah proyek *open source* dengan *Controller platform* yang *modular, pluggable,* dan fleksibel. Kontroler ini diimplementasikan pada perangkat lunak dan ditampung dalam *Java Virtual Machine* (JVM)-nya sendiri. Dengan demikian, OPENDAYLIGHT dapat digunakan pada perangkat keras dan *platform* sistem operasi apapun yang mendukung Java.[7]

*OpenDaylight Controller* dinamai berdasarkan nama-nama elemen pada tabel periodik. Versi pertama dari OPENDAYLIGHT yaitu *Hydrogen* dirilis 10 bulan setelah OPENDAYLIGHT *Project* didirikan. Sekarang, versi kedua yang diberi nama *Helium* telah beredar dengan menawarkan perbaikan-perbaikan berupa:[8]

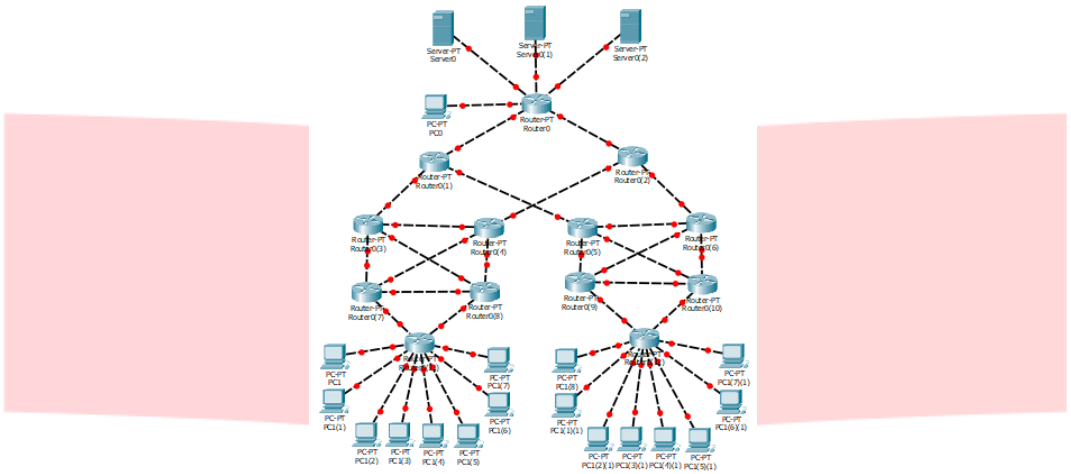
- Peningkatan *clustering* dan *high-availability support* untuk kontroler inti.
- Akuntansi, otorisasi dan autentikasi untuk peningkatan keamanan.
- Antarmuka pengguna yang baru, yaitu *dlux* (*the openDayLight User eXperience*), yang dapat berjalan terpisah dari perangkat lunak OPENDAYLIGHT selama ada koneksi IP.
- Dukungan untuk *Apache Karaf container*, sehingga memungkinkan operator jaringan untuk menyesuaikan instalasi, hanya menggunakan fitur yang mereka perlukan.
- Integrasi lebih dalam dengan *OpenStack*, termasuk perbaikan yang signifikan terhadap proyek *Open vSwitch Database* (OVSDB), dan pratinjau teknologi fitur *OpenStack* seperti *Security Group, Distributed Virtual Router* dan *Load Balancing-as-a-Service*.




Gambar 2.2.Arsitektur OPENDAYLIGHT HELIUM

## 3. Perancangan Sistem

### 3.1.Sistem Secara Umum



*Gambar 3.1.Gambar Topologi Jaringan*

Gambar 3.1. diatas adalah topologi jaringan yang akan dirancang, terdiri dari 19 *host* (16 *client* dan 3 *server*), 13 buah *switch*, dan 1 buah *controller* yang digunakan untuk pengimplementasian sistem. Sistem ini dibagi menjadi beberapa *devices*,
yaitu :

- *Host*, merupakan pengguna/*client* yang menggunakan *service* pada *server*.
- *OpenFlow Switch*, perangkat ini bertugas untuk *forwarding* paket dari *controller*.
- *Controller*, perangkat ini bertugas sebagai *network policy* terhadap jaringan.
- *Server*, bertugas untuk menyediakan layanan terhadap *user* yaitu sebuah *web server.*

### 3.2. Perancangan Load Balancing dengan Metode Round Robin dalam SDN Menggunakan Controller OPENDAYLIGHT

Dalam perancangan *load balancing* dengan metode *round robin* dalam *software defined networking* menggunakan *controller* OPENDAYLIGHT harus memenuhi standar penulisan dan pendeklarasian setiap modul yang ditentukan. OPENDAYLIGHT dapat diimplementasikan dengan *Northbound Interface* sehingga proses pembuatan program aplikasinya lebih sederhana dan aplikatif. *Northbound interface* yang digunakan berasal dari *controller* OPENDAYLIGHT. Berikut ini adalah tahap-tahap logis yang dilakukan untuk mengimplementasikan *load balancing* dengan metode *round robin* dalam *software defined networking* menggunakan *controller* OPENDAYLIGHT, dimana *server load balancer* yang dalam hal ini diterapkan oleh *Switch* melakukan pemetaan terhadap *packet request* yang masuk menuju *server*. Aspek utama dalam proses ini adalah komputer *client* hanya mengetahui alamat IP (IP Address) dari *load balancer* yaitu ‘127.0.0.1’.

1. Tetapkan alamat *controller* sebagai 127.0.0.1
2. Inisialisasi list dari server-server
3. Masuk akses packet-in terhadap IP dari client ‘X’.
    - Tetapkan server ‘Y’ dengan menggunakan metode round-robin
    - Paket masuk yang mengalir (client menuju ke server):
        - Match: Paket yang masuk
        - Penetapan ulang destination_Mac, destination_IP, dari paket yang diterima dari client ‘X’.
        - Teruskan ke Mac_to_port[‘Y’]
    - Paket masuk yang mengalir dari arah sebaliknya (server menuju ke client):
        - Match : Src (IP, MAC, TCP_Port) = Y, Dst = X,
        - Penetapan ulang source_Mac, source_IP menjadi client_IP
        - Teruskan ke Mac_to_port[‘X’]
    - Semua paket request yang berikutnya akan secara langsung dikirim ke server yang terpilih (berdasarkan metode round-robin) tanpa harus melalui controller lagi (karena client sebagai sumber paket request telah diidentifiksasi sebelumnya).

### 3.3.Parameter Pengujian Sistem

Parameter pengujian merupakan parameter yang akan dihitung dalam pengujian sistem *load balancing*.

#### 3.3.1.Response time

Pengujian *response time* bertujuan untuk mengetahui seberapa lama sebuah data berjalan dalam sebuah jaringan dari satu node ke node tujuan. Dimana pada pengujian ini bertujuan untuk menunjukan *response time* terhadap *server*. Perhitungan *response time* dilakukan dengan cara pengiriman ICMP dari *host* ke *server*.

#### 3.3.2.Throughput

Pengujian dengan parameter throughput bertujuan untuk mengetahui kecepatan pengiriman data dalam jaringan. Pengujian akan dilakukan dengan melihat adaptasi *bandwidth* pada *server* dengan *load balancing* dan tanpa *load balancing*.

#### 3.3.3.Network Convergence

Pengujian dengan parameter *network convergence* ditujukan untuk mengetahui lama waktu yang dibutuhkan jaringan untuk berbagi informasi sampai berada dalam kondisi stabil. Pengujian akan dilakukan dengan pengukuran waktu mulai dari *connection up* sampai dengan kondisi jaringan stabil. Data yang akan diukur adalah ketika pemakaian *load balancing* akan dibandingkan dengan data yang tidak memakai *load balancing*.

#### 3.3.4.Overhead

Pengujian *Overhead* dilakukan untuk mengetahui waktu yang dibutuhkan sistem untuk memproses paket yang masuk sampai selesai menjalankan algoritma. Terdapat dua subjek pengukuran *overhead* yaitu lama waktu dan besar paket, pengukuran waktu diukur dari proses *packet in* sampai algoritma selesai dijalankan. Nilai *overhead* akan didapat dari selisih waktu yang didapat.

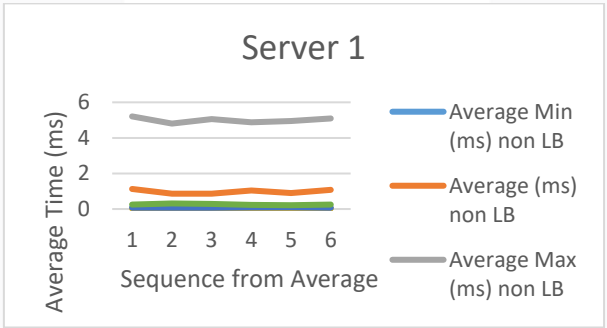
#### 3.3.5.Resource Utilization

Pengujian *resource utilization* dilakukan untuk mengetahui seberapa besar *resource* yang digunakan *controller* ketika pemakaian *load balancing* dan tanpa *load balancing*. Untuk mendapatkan nilai pemakaian memori digunakan utilitas *system info*, *system info* digunakan untuk mengetahui besar penggunaan memori pada saat menjalankan *controller*.

## 4. Pengukuran dan Analisis

### 4.1.Analisis Paket ICMP Sebelum dan Setelah Load-Balancing

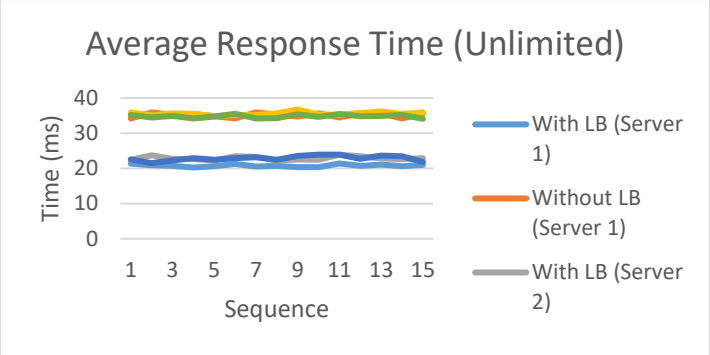
Ketika seluruh client melakukan akses terhadap *server* (melalui paket ICMP) yang tidak menerapkan teknik *load balancing* maka secara pasti besar waktu yang diperoleh untuk melayani seluruh paket tersebut akan lebih banyak atau lama. Dengan metode *round robin* dalam *software defined networking* (SDN) Menggunakan *controller* OPENDAYLIGHT berhasil, maka dilakukan perbandingan *response time* (waktu respon) terhadap seluruh *client* oleh *server* sebelum dan setelah *load balancing* diterapkan.




*Gambar 4.1. Rata-rata dari client ke server 1*

### 4.2.Response Time dengan Bandwidith Unlimited

Pengujian dengan download rate unlimited ditujukan untuk mengetahui perubahan nilai response time pada masing-masing server, pada saat traffic penuh maka response time akan ikut naik seiring dengan penuhnya bandwidth dari server. Pengujian dilakukan dengan 16 host mengirim permintaan paket dengan cara wget file ke server. Proses download akan dilakukan secara berulang.




*Gambar 4.2. Rata-rata response time dengan bandwidth unlimited*

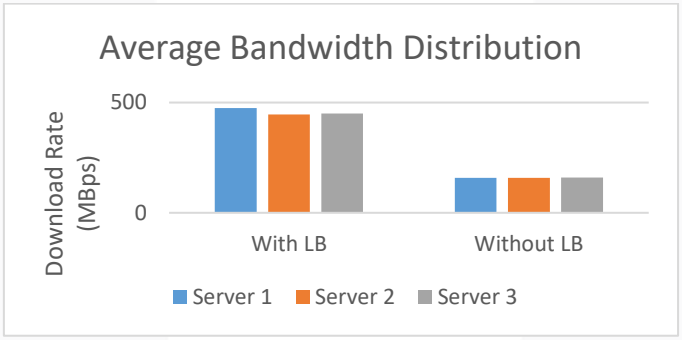
| Server | With LB (ms) | Without LB (ms) |
|---|---|---|
| 1 | 20,79 | 35,14 |
| 2 | 22,9 | 35,57 |
| 3 | 22,87 | 34,81 |

*Tabel 4.1. Nilai rata-rata sebelum dan sesudah load balancing*

Dari hasil pengujian didapatkan rata-rata response pada masing-masing server yaitu pada saat pemakaian load balancing didapat sebesar 20,79 – 22,87 ms sedangkan tanpa load balancing 34,81–35,57 ms. Data yang didapat pada tabel menunjukan bahwa rata-rata response time pada pemakaian load balancing adalah 22,19 ms sedangkan tanpa pemakaian load balancing didapatkan rata-rata yaitu 35,17 ms, disimpulkan dari persentase kenaikan yang didapat bahwa kinerja load balancing dapat menaikan performansi menjadi 58.77%.

**4.3.Bandwidth dengan download rate unlimited**

Pengujian dilakukan dengan melakukan variasi pada *rate download* dimana *rate download* di setting pada kecepatan *unlimited* dan pada *link bandwidth* di *setting* sebesar 50Mb. 16 *host* akan mengunduh *file* dari *server* dan tidak ada batasan kecepatan pada *user*.

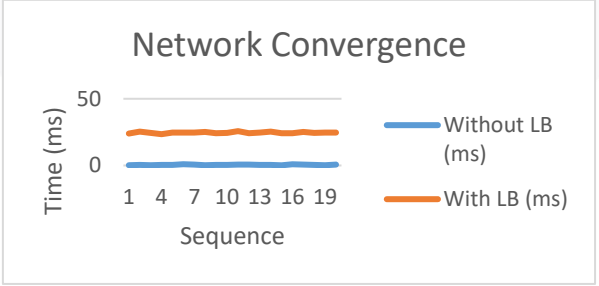



*Gambar 4.3..Rata-rata bandwidth distribution*

pengujian yang telah dilakukan didapatkan bahwa rata kecepatan *download* 16 user pada pemakaian *load balancing* berkisar antara 446,38 – 475,53 KBps karena beban yang diterima *server* dibagi menjadi 3. Data yang didapat bahwa rata-rata *speed* yang diterima *client* saat pemakaian *load balancing* adalah sebesar 457,57 KBps sementara tanpa aplikasi *load balancing* mendapat 158,79 KBps.

**4.4.Pengujian Time Convergence**

Konvergensi merupakan waktu yang dibutuhkan sebuah jaringan untuk stabil saat jaringan pertama kali dijalankan. Pada pengujian ini dilakukan pengujian berupa perhitungan waktu dari connection up pada controller hingga controller siap menerima paket dari host.

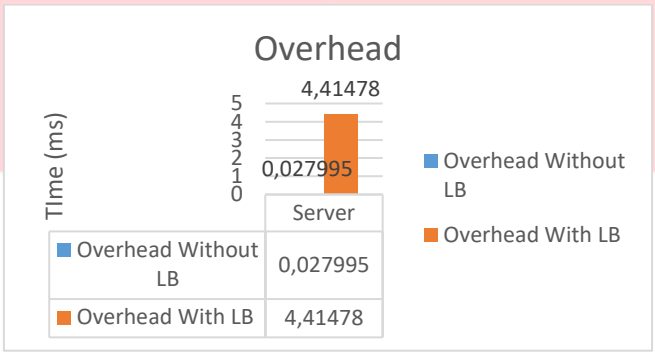


*Gambar 4.4. Network convergence sebelum dan sesudah load balancing*

Gambar diatas merupakan hasil pengujian konvergensi ketika jaringan tidak menggunakan Load balancing dengan yang menggunakan Load balancing. Dari hasil pengujian dihasilkan waktu 0,511 ms untuk sistem tanpa Load balancing dan 24,622 ms untuk sistem dengan Load balancing.

**4.5.Pengujian Overhead**

Pengujian Overhead dilakukan dengan cara yaitu melihat waktu dari packet-in sampai instruksi dikirim ke switch, ketika host baru melakukan request ke controller maka proses tersebut dinamakan packet-in dan mulai dari sini waktu dihitung sampai dengan rule dikirim ke switch.

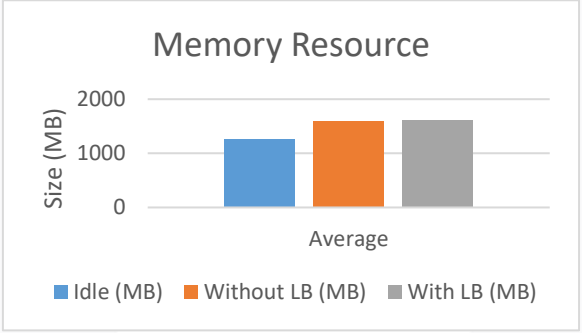



*Gambar 4.5. Overhead sebelum dan sesudah load balancing*

balancing dibutuhkan waktu untuk melihat host dan tujuan server yang ditentukan, baru kemudian dikirim ke server yang ditentukan oleh algoritma.

**4.6.Pengujian Resource Utilization**

Pengujian resource utilization bertujuan untuk melihat sebarapa besar resource memory yang terpakai saat pemakaian load balancing dan tanpa load balancing.




*Gambar 4.6. Resource utilization yang terpakai*

Secara umum pemakaian memori ketika pengujian tidak jauh berbeda, ketika kondisi system idle pemakaian memori berada pada sekitar 1200 MB dan pada tanpa load balancing memori yang dipakai yaitu sekitar 1600 MB sedangkan dengan load balancing yaitu sekitar 1650 MB dari total memori keseluruhan yaitu 2 GB. Dari data yang di dapat, menunjukan bahwa pemakaian resource lebih banyak pada aplikasi balancing, hal ini disebabkan karena terdapat banyak proses proses pertukaran informasi yang dilakukan oleh controller.

## 5. Kesimpulan dan Saran

### 5.1 Kesimpulan

Berikut ini adalah hasil analisis kinerja algoritma round robin pada jaringan SDN :

1. Hasil yang didapatkan berbanding jauh dengan setelah *load balancing* yang nilai rata-rata minimum dan maksimum tidak lebih dari 1 ms karena dengan tanpa *load balancing* proses pengiriman paket membutuhkan waktu yang lebih lama.
2. Hasil yang didapatkan dari uji parameter *response time* dapat disimpulkan dari persentase kenaikan yang didapat bahwa kinerja *load balancing* dapat menaikan performansi menjadi 55%.

3. Hasil dari uji *bandwidth* yaitu didapatkan bahwa kenaikan performansi dari tanpa *load balancing* ke *load balancing* kurang lebih 185%. Dapat disimpulkan bahwa pemakaian *load balancing* dapat memperbaiki kualitas layanan *server*.
4. Hasil dari *network convergence* pada sistem *load balancing* membutuhkan waktu konvergensi lebih lama dikarenakan saat pertama kali *load balancing* tersebut akan melakukan perhitungan yang dibutuhkan untuk menjalankan fungsi *load balancing*.
5. Untuk hasil *overhead* dapat dilihat bahwa nilai setelah menggunakan *load balancing* lebih besar daripada sebelum menggunakan *load balancing*, karena saat melakukan *load balancing* membutuhkan waktu untuk memilih lajur paket dan *server* yang akan dituju. Kemudian untuk *resource utilization* bisa disimpulkan ketika saat keadaan *idle* penggunaan memori lebih sedikit dibandingkan pada saat program *load balancing* dijalankan.

### 5.2 Saran

Pada penelitian ini terdapat beberapa saran dalam pengembangan penelitian ini
lebih lanjut, yakni :

1. Perlunya pengimplementasian SDN dengan perangkat yang sesungguhnya sehingga proses perhitungan response time lebih akurat karena tidak tergantung hanya pada satu komputer.
2. Percobaan dapat menggunakan metode load balancing yang berbeda contohnya Weighted Round-Robin, Least Connections, dan lain-lain.
3. Percobaan dapat dilakukan dengan mengkombinasikan teknik load balancing dengan teknik lainnya seperti firewall, QOS, bandwidth management, dan lain-lain.

## DAFTAR PUSTAKA


[1] R. Wang, D. Butnariu, and J. Rexford, "OpenFlow-Based Server Load Balancing Gone Wild Into the Wild : Core Ideas," *Hot-ICE'11 Proc. 11th USENIX Conf. Hot Top. Manag. internet, cloud, Enterp. networks Serv. Serv.*, p. 12, 2011.

[2] O. N. Fundation, "Software-Defined Networking: The New Norm for Networks [white paper]," *ONF White Pap.*, pp. 1–12, 2012.

[3] G. N. Senthil and S. Ranjani, "Dynamic Load Balancing using Software Defined Networks," *Int. J. Comput. Appl.*, pp. 11–14, 2015.

[4] Y. Han, S. Seo, J. Li, J. Hyun, J.-H. Yoo, and J. W.-K. Hong, "Software defined networking-based traffic engineering for data center networks," *16th Asia-Pacific Netw. Oper. Manag. Symp.*, pp. 1–6, 2014.

[5] W. Braun and M. Menth, "Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices," *Futur. Internet*, vol. 6, no. 2, pp. 302–336, 2014.

[6] N. McKeown *et al.*, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, p. 69, 2008.

[7] R. Kartadie and B. Satya, "Uji Performa Kontroler Floodlight Dan Opendaylight Sebagai Komponen Utama Arsitektur Software-Defined Network," *Semnasteknomedia Online*, pp. 6–8, 2015.

[8] Y. Bayu, A. Pranawa, R. M. Ijtihadie, and W. Wibisono, "Menggunakan Software Defined Network untuk Meningkatkan Network Reliability pada Jaringan," vol. 6, no. 1, 2017.

[9] D. Mithbavkar, H. Joshi, H. Kotak, D. Gajjar, and L. Perigo, "Round Robin Load Balancer using Software Defined Networking ( SDN )," *Capstone Team Res. Proj.*, pp. 1–9, 2016.

[10] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a model-driven SDN controller architecture," *Proceeding IEEE Int. Symp. a World Wireless, Mob. Multimed. Networks 2014, WoWMoM 2014*, 2014.

[11] M. E. Mustafa, "LOAD BALANCING ALGORITHMS ROUND-ROBIN ( RR ), LEAST-CONNECTION , AND LEAST LOADED EFFICIENCY," vol. 1, no. 1, pp. 25–29, 2017.

[12] H. Nasser, "ANALISIS ALGORITMA ROUND ROBIN , LEAST CONNECTION , DAN RATIO PADA LOAD BALANCNG," vol. 12, no. 1, pp. 25–32, 2016.

[13] G. Triono, T. Informasi, S. Tinggi, T. Surabaya, and L. V. Server, "Implementasi Load Balancing Dengan Menggunakan Algoritma Round Robin Pada Kasus," pp. 169–176, 2015.