Implementasi Load Balancing Menggunakan Algoritma Modified Weighted Round Robin-Retrieve Packet Pada Software Defined Networking

Zainal Abidin[1], Tutuk Indriyani[2], Danang Haryo Sulaksono[3]
[1,2,3]Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Adhi Tama Surabaya
Email : welcome.zainalabidin@gmail.com

Abstract *Client's request for traffic problems is so huge that causes the single server difficult in handling the traffic load. Therefore, the system of load balancing is required as it is a technique to equally distribute the traffic load on the two or more connection lines so that the traffic can run optimally. Thus, the load balancing is crucial to implement by using Modified Weighted Round Robin-Retrieve Packet on the Software-Defined Networking. Based on the parameter of average response-time in time limits 0.1, 0.2, and 0.3 seconds, the scores were 0.016-0.04, 0.02-0.04, and 0.014-0.032 seconds consecutively. Based on the parameter of data transaction per second in time limits 0.1; 0.2, and 0.3 seconds, the scores respectively were 49.614-111.306, 41.678-107.032, and 37.806-102.84 data transaction/second.*

Keywords: Load balancing, average response-time, data transaction per second, Software Defined Networking, Modified Weighted Round Robin-Retrieve Packet.

Abstrak Banyaknya penanganan sejumlah besar lalu lintas dalam melayani *request* dari client, akan menyulitkan *Server* tunggal dalam menangani beban sebesar itu. Maka diperlukannya sistem *load balancing* yaitu teknik untuk mendistribusikan beban *traffic* pada 2 atau lebih jalur koneksi secara seimbang, agar *traffic* dapat berjalan optimal. Sehingga diimplementasikan *load balancing* menggunakan algoritma *Modified Weighted Round Robin-Retrieve Packet* pada *Software Defined Networking*. Berdasarkan parameter Waktu Respon Rata-rata dengan *time limit* 0.1 detik, diperoleh nilai antara 0.016 – 0.04 detik, ketika *time limit* diperbesar menjadi 0.2 detik, diperoleh nilai antara 0.02 – 0.04 detik, ketika *time limit* diperbesar menjadi 0.3 detik diperoleh nilai antara 0.014 – 0.032 detik. Berdasarkan parameter Transaksi Data per Detik dengan *time limit* 0.1 detik diperoleh nilai antara 49.614 – 111.306 transaksi data per detik, ketika *time limit* diperbesar menjadi 0.2 detik diperoleh nilai antara 41.678 – 107.032 transaksi data per detik, dan ketika *time limit* diperbesar menjadi 0.3 detik diperoleh nilai antara 37.806 – 102.84 transaksi data per detik.

Kata kunci: *Load balancing*, Waktu Respon Rata-rata,Transaksi Data per Detik, *Software Defined Networking, Modified Weighted Round Robin-Retrieve Packet.*

1. Pendahuluan

Dalam arsitektur jaringan tradisional yang hanya mengandalkan konektivitas melalui pemasangan peranti perangkat keras jaringan seperti *switch* dan *hub*, hal ini memiliki beberapa keterbatasan karena cara kerja aplikasi *modern* telah berubah, *Network Policy* yang tidak konsisten, serta adanya ketergantungan pada *vendor* peranti jaringan (US:*Open Networking Foundation*. 2012). Sehingga dari berbagai keterbatasan arsitektur jaringan tradisional tersebut berkembang sebuah pendekatan dalam jaringan yang bernama *Software Defined Networking* (SDN) yaitu sebuah pendekatan arsitektur jaringan komputer yang sangat fleksibel karena dapat dikonfigurasi dan dikendalikan melalui *software* terpusat yang memungkinkan *administrator* sistem untuk mempercepat koneksi penyediaan jaringan, memiliki kontrol pusat pada sebuah program lalu lintas jaringan tanpa memerlukan akses fisik ke perangkat keras jaringan, serta tidak perlu bergantung pada *vendor* atau produk tertentu di dalam implementasi jaringan.

Salah satu permasalahan pada jaringan sekarang ini adalah banyaknya penanganan sejumlah besar lalu lintas dalam melayani banyak *Client*. Sehingga sulit bagi *Server* tunggal untuk menangani beban sebesar itu. Maka diperlukannya sistem *Load Balancing* yaitu teknik untuk mendistribusikan beban *traffic* pada 2 atau lebih jalur koneksi secara seimbang, agar *traffic* dapat berjalan optimal, memperkecil waktu respon, dan meningkatkan jumlah transaksi data per detik. *Load Balancer* pada *Software Defined Networking* tidak perlu *hardware* secara khusus. *Load Balancer* pada SDN dapat diprogram dan memungkinkan untuk merancang dan menerapkan strategi *Load Balancing* sendiri. Umumnya, sistem pembuat keputusan kemana arus data akan dikirimkan dibuat menyatu dengan perangkat kerasnya, sedangkan SDN memungkinkan penggunanya untuk mengelola layanan jaringan termasuk *Load Balancing, Routing, Acces Control, Multicast*, dan tugas-tugas rekayasa lalu lintas jaringan lainnya melalui *software* terpusat tanpa harus konfigurasi secara langsung pada perangkat keras (Khondoker, Zaalouk, Marx dan Bayarou. 2014).

Dalam pengimplementasian *Load Balancing* diperlukan adanya algoritma penjadwalan yang dijalankan, sebagaimana pada penelitian sebelumnya yang menggunakan algoritma *Weighted Round Robin* sebagai algoritma penjadwalan, terdapat beberapa keterbatasan pada algoritma tersebut dalam hal layanan jumlah paket yang memiliki prioritas rendah sehingga mengakibatkan *delay* dan *jitter* (Khondoker, Zaalouk, Marx dan Bayarou. 2014), sehingga hal ini berpengaruh negatif pada jumlah rata-rata waktu respon server dan jumlah transaksi data per detik, maka dari keterbatasan tersebut diusulkan algoritma *Modified Weighted Round Robin-Retrieve Packet* yang diharapkan dapat mengurangi *delay* sehingga bisa memperkecil waktu respon, dan meningkatkan jumlah transaksi data per detik (Mardini dan Abu Alfoul. 2011).

Studi ini bertujuan untuk mengimplementasikan *Load Balancing* menggunakan algoritma *Modified Weighted Round Robin-Retrieve Packet* pada *Software Defined Networking* dalam hal kemampuan algoritma tersebut untuk meningkatkan atau memperbaiki kinerja algoritma penjadwalan *Weighted Round Robin* pada penelitian sebelumnya yang terkait dalam *Software Defined Networking*, seperti memperkecil rata-rata waktu respon dan meningkatkan jumlah transaksi data per detik.

2. Tinjauan Pustaka

2.1 *Software Defined Networking*

Software Defined Networking (SDN) merupakan sebuah pendekatan arsitektur jaringan komputer yang memisahkan *control plane* dari sebuah perangkat jaringan komputer (*switch* atau *router*) dengan *data plane* perangkat jaringan komputer tersebut (Khondoker, Zaalouk, Marx dan Bayarou. 2014). Pemisahan *data plane* dan *control plane* ini memungkinkan untuk memprogram perangkat tersebut sesuai dengan yang diiinginkan secara terpusat (SDN *Controller*), sehingga hal ini memungkinkan untuk mengontrol, memonitor, dan mengatur sebuah jaringan komputer dari sebuah titik (*node*) terpusat tersebut.

2.2 Load Balancing

Load-balancing mendistribusikan beban kerja pada dua atau lebih komputer, *link* jaringan, CPU, *hard drive*, atau sumber daya lainnya, untuk mendapatkan pemanfaatan sumber daya yang optimal (US: FORTINET. 2014).

2.2.1 Algoritma *Modified Weighted Round Robin-Retrieve Packet*

Algoritma *Modified Weighted Round Robin-Retrieve Packet* adalah algoritma yang penulis gunakan untuk penelitian dimana diharapkan nantinya dapat meningkatkan performansi algoritma penjadwalan sebelumnya yaitu *Weighted Round Robin*. Sebuah beban untuk setiap antrian ditentukan berdasarkan prioritas data pada setiap antrian. Beban ini memungkinkan *user* untuk mengirim sebuah *quantum packet* di dalam putaran layanan yang aktif. Dan proses ini akan pindah ke proses selanjutnya setelah semua antrian telah dilayani. Di dalam algoritma ini tetap menggunakan fungsi GCD (*Greates Common Divisor*) untuk mengevaluasi *counter* beban, dan mengembalikan suatu nilai yang digunakan untuk penghitungan *counter* beban tersebut. Proses akan berlanjut melayani *packet* sampai semua antrian kosong dan *counter* beban bernilai 0 (Mardini dan Abu Alfoul. 2011).

Putaran layanan yang kecil sehingga dibutuhkan perhitungan di setiap awal putaran layanan, sehingga algoritma ini dapat memaksimalkan putaran layanan untuk meningkatkan performansi dalam hal penjaminan jumlah *packet* yang dilayani akan lebih tinggi dibandingkan algoritma sebelumnya,

sehingga dilakukan perkalian dengan sebuah nilai konstan untuk memaksimalkan putaran layanan, dimana nilai konstan tersebut tergantung pada ukuran jaringan, semakin besar jaringan maka nilai konstan semakin kecil dan sebaliknya.

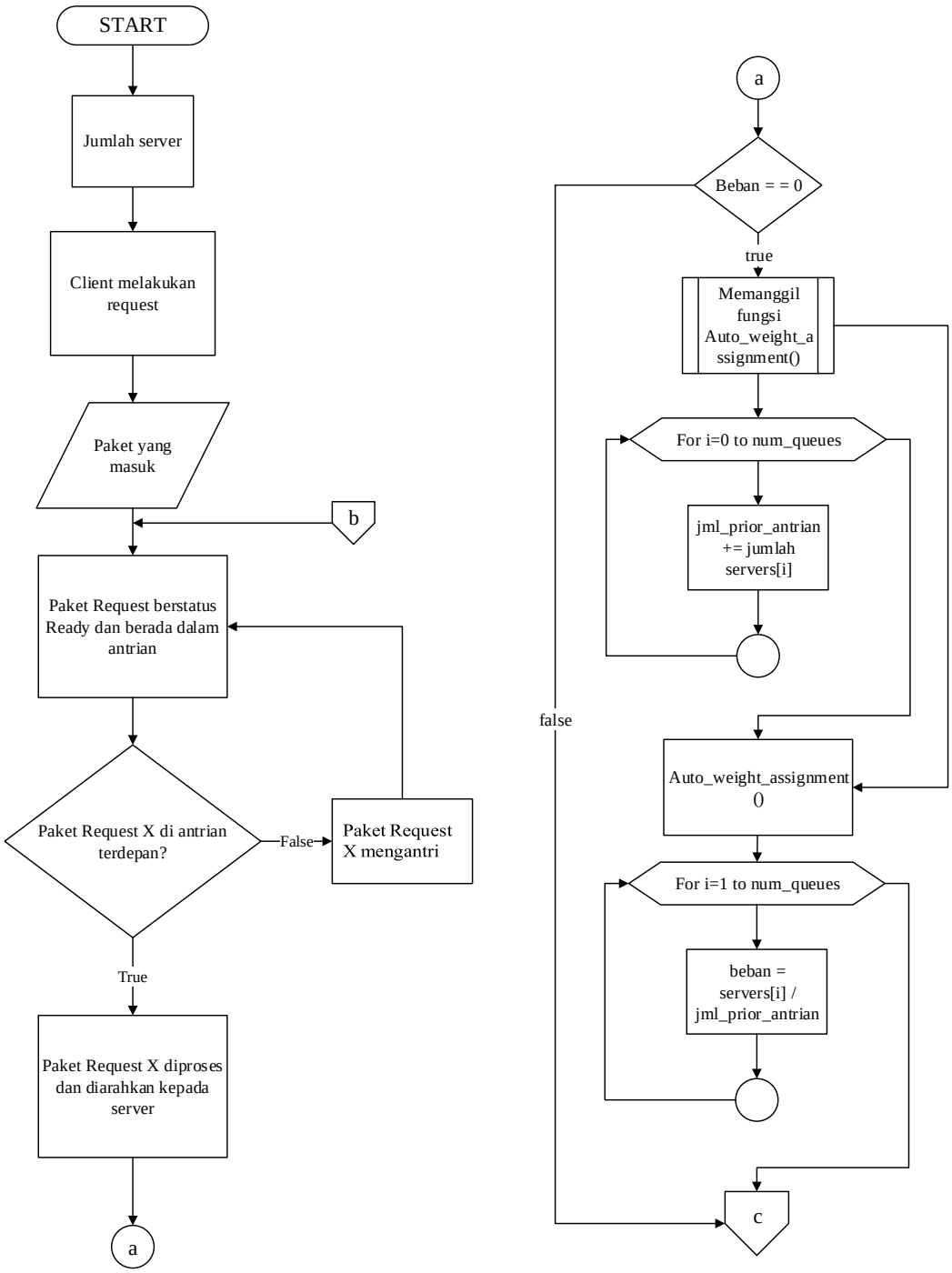
3. Perancangan Sistem

3.1 Algoritma *Modified Weighted Round Robin-Retrieve Packet*

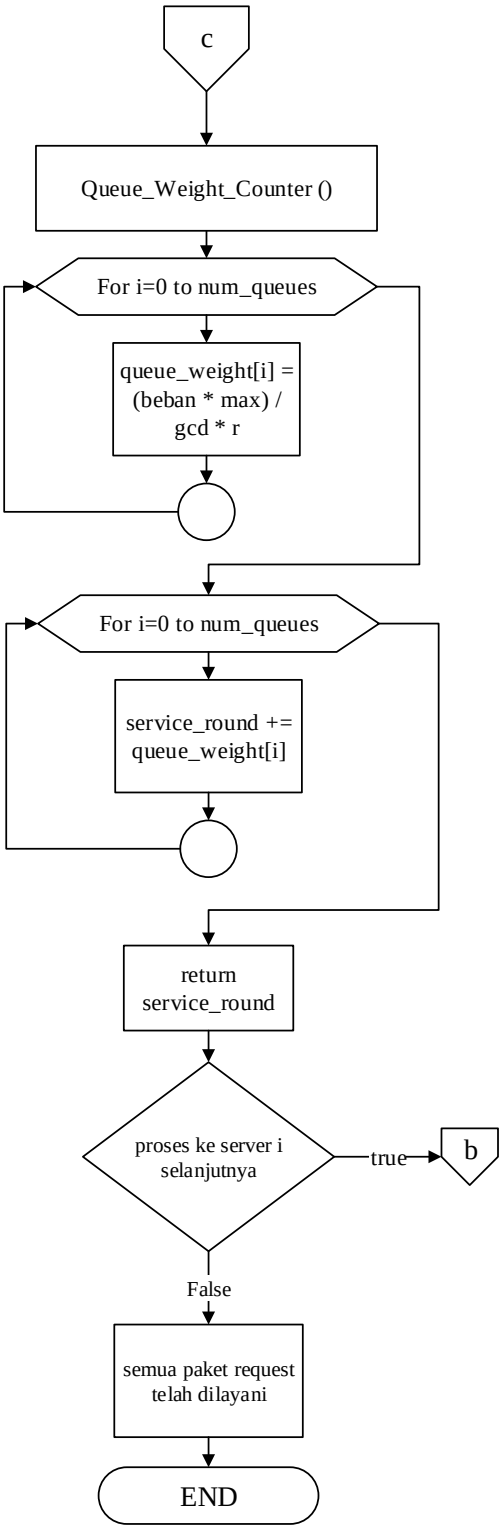
Beberapa alasan mengapa memerlukan modifikasi pada algoritma *Weighted Round Robin* adalah untuk meningkatkan jumlah *service round*, karena ketika *service round* yang merupakan jumlah dari semua nilai counter beban antrian itu kecil akan berpengaruh pada jumlah transaksi per detik, semakin kecil dan terbatasnya jumlah *service round* maka semakin kecil pula jumlah transaksi layanan paket yang bisa dilayani oleh *server*, sehingga diharapkan nantinya juga dapat memperkecil waktu respon *server*.

Kemudian salah satu hal mendasar pada modifikasi tersebut adalah adanya fungsi *Retrieve Packet* yang berfungsi untuk pengecekan beban pada setiap antrian serta jika kondisi beban tidak *assigned* (ditugaskan) pada antrian nantinya ia akan menjalankan fungsi *auto weight assignment* yang memberikan beban pada setiap antrian berdasarkan prioritas dan memunginkan paket untuk dilayani pada *service round* yang sama hingga proses berpindah setelah putaran prioritas berakhir. Jika tidak ada pengecekan beban pada setiap antrian maka pemilihan *server* akan sesuai *service round* berdasarkan nilai beban maksimum yang ditentukan. Hal ini membuat *service round* menjadi terbatas dalam pelayanan paket pada putaran layanan yang sama.

ISSN: 2579-566X (Online)
ISSN: 2477-5274 (Print)



Gambar 3.5 : ***Flowchart*** **penerapan Algoritma** ***Load Balancing*** **menggunakan** ***Modified Weighted Round Robin-Retrieve Packet***



Gambar 3.6 : ***Flowchart*** **lanjutan penerapan Algoritma** ***Load Balancing*** **menggunakan** ***Modified Weighted Round Robin-Retrieve Packet***

ISSN: 2579-566X (Online)
ISSN: 2477-5274 (Print)

Seperti terlihat pada gambar 3.5 dan 3.6 *flowchart* tersebut menggambarkan proses algoritma *Modified Weighted Round Robin-Retrieve Packet* yang dirancang ke dalam *Controller* POX pada *Software Defined Networking* :

1. Tentukan jumlah *server*
2. *Client* melakukan *request*
3. Paket yang masuk
4. Paket *Request* berstatus *Ready* dan berada dalam antrian
5. Paket *Request* X di antrian terdepan?
   - jika iya, maka Paket *Request* X diproses dan diarahkan ke *server*
   - jika tidak, Paket *Request* X mengantri ke proses 4
6. *Server* =
   - Menjalankan Fungsi *Retrieve*
   - jika beban = 0, maka beban = jumlah *server* / jumlah antrian
   - dihitung jumlah semua antrian aktif
     jumlah antrian = jumlah *server*
   - Menjalankan *Counter* Penjadwalan
     beban antrian ke i = (beban * *max*) / gcd_s * r
     kembali ke *server* i
7. Apakah proses ke *server* i selanjutnya,
   - jika iya, maka kembali ke proses 4
   - jika tidak, maka semua paket *request* telah dilayani
8. Selesai

4. Implementasi Sistem

4.1 Instalasi dan konfigurasi unit

Pada tahapan ini dilakukan beberapa instalasi program serta konfigurasi yang diperlukan untuk proses menjalankan sistem *Load Balancing* pada *Software Defined Networking.*

4.2 Pembuatan Program

Pada tahapan ini dilakukan pembuatan kode program untuk POX *Controller* serta kode program yang mengimplementasikan algoritma *Modified Weighted Round Robin-Retrieve Packet.*

4.3 Uji integrasi PC *Openflow Switch* dengan PC *Controller*

Gambar 4.3 : Tampilan integrasi PC Openflow Switch dengan PC Controller

Pada gambar 4.3 di atas merupakan uji integrasi antara unit *Openflow Switch* dengan unit *Controller*, dimana mula-mula dilakukan penentuan *datapath-id* yang didapatkan dari *mac address Ethernet* yang dipilih untuk jalur koneksi pada *Controller* yaitu *Ethernet* 1 dengan IP *Address* 10.10.2.1 dengan *mac address* 001ec95810ac, kemudian dijalankan modul untuk protokol *openflow* dengan mengalamatkan pada ip *address* milik *controller* yaitu 10.10.2.2 dengan *default port* 6633, kemudian terlihat status koneksi berhasil terhubung.

```
root@Controller:~/pox-eel/pox# ./pox.py pox.misc.OFmwrr-rp
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.5.0 (eel) is up.
```

Gambar 4.4 : Tampilan PC Controller berhasil menjalankan program dan terintegrasi dengan PC Openflow Switch

Pada gambar 4.4 di atas menunjukkan bahwa uji program *load balancer* dengan modul *Openflow* yang dijalankan pada program POX *Controller* berhasil berjalan tanpa *error* serta menunjukkan integrasi dengan unit *Openflow Switch* telah berhasil terhubung dengan *datapath-id* dari unit *Openflow Switch* yang sudah ditentukan.

4.2.2 Uji program *Load Balancer* menggunakan algoritma *Modified Weighted Round Robin-Retrieve Packet* menggunakan *Siege Load Tester* pada PC *Client.*

Pengujian menggunakan data yang telah ditentukan yaitu *file* berformat .iso berukuran 973 MB untuk menguji performansi setelah dilakukan proses *Load Balancing* dengan uji performansi parameter Waktu Respon Rata-rata dan Transaksi Data per Detik berdasarkan jumlah (*virtual concurrent user request* dan *time limit*) yang diinginkan secara simultan, dimana pada proses ini dilakukan analisa hasil untuk parameter pengujian dari algoritma *Modified Weighted Round Robin-Retrieve Packet*. Dimana pengujian nantinya dilakukan 5 kali untuk setiap variasi *request*, dimana setiap *virtual user* mengirimkan *request* sebanyak 10, dengan 3 variasi *time limit* yaitu 0.1, 0.2, dan 0.3 detik, sehingga total pengujian beban sistem *Load Balancing* menggunakan algoritma *Modified Weighted Round Robin-Retrieve Packet* pada *Software Defined Networking* adalah 150 kali pengujian.

a. Pengujian pertama hingga kelima menggunakan variasi *request* 130, 260, 390, 520, dan 650, *virtual concurrent users* dengan *time limit* 0.1 detik.
b. Pengujian pertama hingga kelima menggunakan variasi *request* 130, 260, 390, 520, dan 650, *virtual concurrent users* dengan *time limit* 0.2 detik.
c. Pengujian pertama hingga kelima menggunakan variasi *request* 130, 260, 390, 520, dan 650, *virtual concurrent users* dengan *time limit* 0.3 detik.

4.1 Evaluasi Hasil

Berikut merupakan evaluasi hasil pengujian yang didapat dari pengujian performansi *web* dengan beban uji dari data *dummy* yang diakses melalui alamat pada *Web E-Learning* milik *server*.

1. Hasil rata-rata dari pengujian berdasarkan *time limit* 0.1 detik untuk variasi *request* 130, 260, 390, 520, dan 650, didapatkan hasil untuk kedua parameter ditunjukkan pada tabel dan grafik di bawah ini.

Tabel 4.6 Performansi *Web* dengan *time limit* 0.1 detik

No.	Request	Time Limit (detik)	Waktu Respon Rata-rata (detik)	Total Transaksi Data (transaksi/detik)
1.	130	0.1	0.016	49.614
2.	260	0.1	0.02	81.784
3.	390	0.1	0.026	97.538
4.	520	0.1	0.03	105.066
5.	650	0.1	0.04	111.306

Pada tabel 4.6 di atas merupakan hasil rata-rata untuk performansi *web* dari pengujian pertama hingga keSlima berdasarkan parameter Waktu Respon Rata-rata dan Transaksi Data per Detik dengan *time limit* 0.1 detik , terlihat bahwa hasil untuk parameter Waktu Respon Rata-rata dari variasi *request*

130 hingga 650 didapatkan nilai yang semakin besar, dimana terlihat semakin besar jumlah *request* maka semakin besar nilai waktu respon yang dibutuhkan, namun secara keseluruhan didapatkan hasil waktu respon untuk variasi *request* ini cukup kecil karena maksimal nilai yang dihasilkan adalah 0.04 detik untuk variasi request terbesar, kemudian untuk parameter Transaksi Data per Detik, terlihat bahwa nilai yang didapatkan semakin besar berbanding lurus dengan semakin besarnya jumlah *request*, dimana besaran nilai maksimal dari request menghasilkan transaksi data hingga 111.306 transaksi data per detiknya, hal ini menunjukkan transaksi data per detik berhasil meningkat walaupun dengan jumlah *request* yang semakin besar.

2. Hasil rata-rata dari pengujian berdasarkan *time limit* 0.2 detik untuk variasi *request* 130, 260, 390, 520, dan 650, didapatkan hasil untuk kedua parameter ditunjukkan pada tabel di bawah ini.

Tabel 4.12 Performansi *Web* dengan *time limit* 0.2 detik

No.	Request	Time Limit (detik)	Waktu Respon Rata-rata (detik)	Total Transaksi Data (transaksi/detik)
1.	130	0.2	0.02	41.678
2.	260	0.2	0.02	71.594
3.	390	0.2	0.024	88.482
4.	520	0.2	0.03	100.264
5.	650	0.2	0.04	107.032

Pada tabel 4.12 di atas merupakan hasil rata-rata untuk performansi *web* dari pengujian pertama hingga kelima berdasarkan parameter Waktu Respon Rata-rata dan Transaksi Data per Detik dengan *time limit* 0.2 detik, terlihat bahwa hasil untuk parameter Waktu Respon Rata-rata dari variasi *request* 130 dan 260 didapatkan nilai yang sama yaitu 0.02 detik, namun pada variasi *request* 390 hingga 650 didapatkan nilai yang semakin besar, namun secara keseluruhan didapatkan hasil waktu respon untuk variasi *request* ini cukup kecil karena maksimal nilai yang dihasilkan adalah 0.04 detik sebagaimana nilai pada variasi *time limit* 0.1 detik sebelumnya untuk variasi *request* terbesar, kemudian untuk parameter Transaksi Data per Detik, terlihat bahwa nilai yang didapatkan semakin besar berbanding lurus dengan semakin besarnya jumlah *request*, dimana besaran nilai maksimal dari *request* menghasilkan transaksi data 107.032 transaksi data per detiknya, hal ini menunjukkan transaksi data per detik berhasil meningkat walaupun dengan jumlah *request* yang semakin besar.

Tabel 4.18 Performansi *Web* dengan *time limit* 0.3 detik

No.	Request	Time Limit (detik)	Waktu Respon Rata-rata (detik)	Total Transaksi Data (transaksi/detik)
1.	130	0.3	0.014	37.806
2.	260	0.3	0.02	62.002
3.	390	0.3	0.022	81.026
4.	520	0.3	0.03	94.754
5.	650	0.3	0.032	102.84

Pada tabel 4.18 di atas merupakan hasil rata-rata untuk performansi *web* dari pengujian pertama hingga kelima berdasarkan parameter Waktu Respon Rata-rata dan Transaksi Data per Detik dengan *time limit* 0.3 detik , terlihat bahwa hasil untuk parameter Waktu Respon Rata-rata dari variasi *request* 130 hingga 650 didapatkan nilai yang semakin besar, dimana terlihat semakin besar jumlah *request* maka semakin besar nilai waktu respon yang dibutuhkan, namun secara keseluruhan didapatkan hasil waktu respon maksimal untuk variasi *request* ini lebih kecil dibandingkan dua variasi *time limit* sebelumnya dimana dengan variasi *request* terbesar didapatkan nilai maksimal sebesar 0.032 detik, kemudian untuk parameter Transaksi Data per Detik, terlihat bahwa nilai yang didapatkan semakin besar berbanding lurus dengan semakin besarnya jumlah *request*, dimana besaran nilai maksimal dari *request* menghasilkan transaksi data hingga 102.84 transaksi data per detiknya, hal ini

menunjukkan transaksi data per detik berhasil meningkat walaupun dengan jumlah *request* yang semakin besar.

5. Penutup

5.1 Kesimpulan

Berdasarkan hasil dari keseluruhan sistem yang telah dirancang, dianalisis, dan diimplementasikan, maka dapat ditarik beberapa kesimpulan diantaranya yaitu :

1. Pemisahan *data-plane* dan *control-plane* pada perangkat jaringan komputer seperti *router* dan *switch* memungkinkan untuk memprogram perangkat tersebut sesuai dengan yang diinginkan secara terpusat melalui *Controller* SDN (*Software Defined Networking*) dalam hal ini dibuktikan dengan berhasilnya implementasi *Load Balancing* menggunakan POX *Controller* yang diintegrasikan dengan *Openflow Switch* yang diterapkan menggunakan algoritma *Modified Weighted Round Robin-Retrieve Packet*.
2. Berdasarkan hasil 150 kali pengujian sistem untuk variasi *request* 130, 260, 390, 520, dan 650 berdasarkan parameter Waktu Respon Rata-rata dengan variasi *time limit* 0.1 detik, dapat diketahui grafik performansi *web* dengan perolehan nilai antara 0.016 – 0.04 detik, ketika variasi *time limit* diperbesar menjadi 0.2 detik, didapatkan perolehan nilai antara 0.02 – 0.04 detik, kemudian ketika variasi *time limit* diperbesar lagi menjadi 0.3 detik diperoleh nilai antara 0.014 – 0.032 detik. Maka dapat disimpulkan secara keseluruhan untuk perolehan nilai parameter Waktu Respon Rata-rata bahwa perolehan nilai masih sangat kecil dimana nilai maksimalnya hanya 0.04 detik, sehingga hasil uji untuk parameter ini berhasil.
3. Sedangkan berdasarkan hasil 150 kali pengujian sistem untuk variasi *request* 130, 260, 390, 520, dan 650 berdasarkan parameter Transaksi Data per Detik dengan variasi *time limit* 0.1 detik dapat terlihat pada grafik performansi *web* didapatkan perolehan nilai antara 49.614 – 111.306 transaksi data per detik, ketika variasi *time limit* diperbesar menjadi 0.2 detik didapatkan perolehan nilai antara 41.678 – 107.032 transaksi data per detik, dan ketika variasi *time limit* diperbesar lagi menjadi 0.3 detik didapatkan perolehan nilai antara 37.806 – 102.84 transaksi data per detik. Maka dapat disimpulkan secara keseluruhan untuk perolehan nilai parameter Transaksi Data per Detik bahwa perolehan berhasil meningkat seiring bertambahnya jumlah *request* dari masing-masing variasi *time limit*, sehingga hasil uji untuk parameter ini berhasil.

5.2 Saran Pengembangan

Pengembangan ke depan pada sistem *Load Balancing* pada *Software Defined Networking* ini agar menjadi lebih baik dan kompleks, maka dibutuhkanlah saran-saran dalam membangun pengembangan sistemnya. Berikut merupakan saran untuk pengembangan diantaranya yaitu :

1. Dilakukan implementasi dengan berbagai algoritma *load balancing* yang ada untuk lebih meningkatkan hasil dari algoritma *Modified Weighted Round Robin-Retrieve Packet* dari segi parameter yang sama maupun yang lain.
2. Implementasi dapat dilakukan dengan mengkombinasikan antara teknik *Load Balancing* dengan teknik *network policy* yang lain seperti *firewall*, *Bandwidth Management*, *Intrusion Detection System*, dan lain-lain.
3. Penambahan jumlah unit *server* agar dapat menjadikan implementasi *Load Balancing* lebih optimal.
4. Penambahan *security* pada unit *Load Balancer* dalam hal ini *Controller* maupun *Openflow Switch* untuk keamanan ketika ada yang mencoba merusak data-data pada *Web Server*.

DAFTAR PUSTAKA

Khondoker , R., Zaalouk, A., Marx, R dan Bayarou, K. (2014), "*Feature-based Comparison and Selection of Software Defined Networking* (SDN) *Controllers*", World Congress on Computer Applications and Information Systems (WCCAIS), Hammamet, 2014, pp. 1-7. doi: 10.1109/WCCAIS.2014.6916572

ISSN: 2579-566X (Online)
ISSN: 2477-5274 (Print)

Mardini, W., dan Abu Alfoul, M. M. (2011, July 24), "*Modified WRR Scheduling Algorithm for WiMAX*", Network Protocols and Algorithms, *3*, 24-53. doi:10.5296/ npa.v3i2.751, ISSN : 1943-3581

US: Open Networking Foundation. (2012), "*Openflow Switch Specification*", ONF White Paper. Palo Alto.

US: Fortinet Technologies Inc. (2014), "*What's New for FortiOS 5.2",* FortiOS Handbook.