Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 3, No. 7, Juli 2019, hlm. 7078-7084 http://j-ptiik.ub.ac.id

Optimasi *Flow Entries* Untuk Mencegah *Flow Table Overflow* Pada *Server Load Balancing* Berbasis *Software Defined Networking*

Agung Wahyu Setio Budi[1], Achmad Basuki[2]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]agungwahyusb@student.ub.ac.id, [2]abazh@ub.ac.id

Abstrak

Konsep dasar SDN adalah manajemen jaringan yang lebih efisien dengan memisahkan *control plane* dan *data plane*. Salah satu protokol yang menunjang konsep SDN adalah OpenFlow. Permasalahan tentang keterbatasan kemampuan OpenFlow *flow table* dalam menyimpan *flow entries* menjadi salah satu motivasi untuk mencegah terjadinya *flow table overflow*, karena sebagian *switch* yang kompatibel dengan protokol OpenFlow dilengkapi dengan kapasitas *flow table* yang terbatas. Penelitian ini melakukan implementasi *flow removal* untuk mencegah *flow table overflow*, yaitu mekanisme *flow expiry* dan *flow modification*. Implementasi mekanisme *flow removal* dilakukan pada sistem *stateless server load balancing* berbasis *software defined networking*, yang menyimpan *flow entries* di dalam *flow table* untuk tujuan penggunaan kembali sebuah *flow entry*. Sistem *stateless server load balancing* mengacu kepada *flow state*, sehingga dapat memicu terjadinya *flow table overflow*. Dari hasil pengujian, diperoleh hasil bahwa sistem mampu mencegah *flow table overflow*. Sebelum mengimplementasikan mekanisme *flow removal*, 53% dari jumlah paket tidak mendapatkan respon dari *server*. Setelah mengimplementasikan mekanisme *flow removal*, 100% paket dapat terfasilitasi untuk menempati *flow table* dan berhasil mendapatkan pelayanan dari *server*. Selain itu, penggunaan kembali *flow entry* dapat memangkas nilai *network latency*.

Kata kunci: *software defined network, openflow, flow table overflow, flow removal, server load balancing*

Abstract

Software Defined Networking (SDN) is new paradigm in computer network management to replace a complex conventional network into the flexible and efficient network by separating the control plane and data plane.This concept lead the control plane define each of flow to be written into data plane. However, the space in the flow table is limited resource, requires careful management to prevent flow table overflow. This research discusses a flow entries management to prevent flow table overflow in SDN by proposing two mechanism of flow removal, that is flow expiry and flow modification. The flow expiry remove an expired flow based on its time, meanwhile the flow modification will be triggered whenever the flow tablespace is nearly full. The implementation of our proposal carried out on a stateless server load-balancing application on SDN. The evaluation results show that the system is able to prevent flow table overflow. Before implements our proposal, 53% of total packet did not get response from server. After implements our proposal, 100% of total packet can accommodate by flow table and get a service from server. In addition, the reuse of flow entries can reduce the value of network latency.

Keywords: *software defined network, openflow, flow table overflow, flow removal, server load balancing*

1. PENDAHULUAN

Software Defined Network (SDN) dan OpenFlow yang dikembangkan untuk menciptakan jaringan inovatif dan efisien tidak sepenuhnya dapat menyelesaikan permasalahan pada jaringan tradisional. Arsitektur SDN yang memisahkan antara *control plane* dan *data plane* menimbulkan permasalahan baru tentang keterbatasan *flow table* dari OpenFlow *switch* dalam menyimpan *flow entries*. Dalam sebuah penelitian yang berjudul *Openflow Timeouts Demystified*, Adam Zarek mengatakan bahwa hal yang tidak kalah penting dan perlu dilakukan selain mengatur *flow table* adalah mengatur kapan dan bagaimana menghapus *flow entry*, karena sebagian besar *switch* yang kompatibel dengan protokol OpenFlow dilengkapi dengan

ruang *flow table* yang terbatas (Zarek, 2012). Tidak hanya itu, Yang dan Riley juga mengatakan dalam penelitiannya yang berjudul *Machine Learning Based Proactive Flow Entry Deletion for OpenFlow* bahwa kapasitas dari *flow table* terbatas, karena menggunakan *Ternary Content Addressable Memory* (TCAM) yang memiliki konsumsi daya tinggi, biaya *chip* yang mahal, serta kendala *silicon area* (Yang & Riley, 2018).

Sebagai *forwarding function* yang bertugas meneruskan paket, *data plane* mengacu kepada *flow state* dari *flow entries* yang disimpan di dalam *flow table*. Ketika *data plane* mengacu kepada *flow state*, maka suatu saat terdapat kemungkinan bahwa jumlah *flow entries* yang ada di dalam *flow table* akan mencapai batas maksimalnya. Sedangkan keterbatasan kemampuan *flow table* dalam menyimpan *flow entries* dapat menyebabkan *flow table overflow*. *Flow table overflow* merupakan sebuah kondisi dimana *flow entries* telah mencapai batas maksimalnya, sehingga *active flows* harus digusur untuk dapat mengakomodasi *flow entry* yang baru. Kondisi *flow table overflow* dapat menyebabkan beban *controller* meningkat, karena mengharuskan untuk berulang kali memperbarui *flow table*. Dengan meningkatnya beban *controller* dapat mengganggu aliran yang ada, sehingga menyebabkan menurunnya kinerja jaringan (Guo, et al., 2018).

Untuk menghindari *flow table overflow*, maka diperlukan sebuah langkah untuk melakukan manajemen *flow entries* yang ada di dalam *flow table*. Dalam teori OpenFlow *specification*, metode yang digunakan untuk manajemen *flow table* berupa penghapusan *flow entries* disebut mekanisme *flow removal*. Mekanisme *flow removal* sendiri dapat dibagi menjadi 2, pertama adalah *flow expiry* dan yang kedua adalah *flow modification*. Mekanisme *flow expiry* akan berjalan dengan cara menghapus *flow entry* ketika nilai *timeouts* yang diberikan telah berakhir. Sedangkan mekanisme *flow modification* akan berjalan ketika *switch* mengirimkan pesan kontrol eksplisit kepada *controller* untuk melakukan penghapusan terhadap suatu *flow entry*.

Dari beberapa penelitian yang telah dibahas di paragraf sebelumnya, permasalahan keterbatasan *flow table* telah banyak mengundang peneliti untuk menemukan solusinya. Adam Zarek mengatakan dalam penelitiannya yang berjudul *Openflow Timeouts Demystified* bahwa metode *flow timeouts*, baik *idle timeouts* maupun *hard timeouts* untuk mengatur *flow expiry* lebih efektif dan *scalable* daripada mengandalkan sepenuhnya pada *flow modification message*. Namun Adam Zarek juga mengusulkan untuk melakukan kombinasi antara kedua metode *flow removal* tersebut (Zarek, 2012). Selain meneliti tentang *timeouts*, penelitian ini juga menggunakan algoritme FIFO dan *random replacement* untuk menentukan *flow entry* mana yang akan dihapus menggunakan metode *flow modification*. Meskipun dalam penelitian ini mengimplementasikan *hybrid flow table management* atau menggunakan mekanisme *flow expiry* dan *flow modification*, diperoleh hasil bahwa belum ada mekanisme yang efisien untuk melakukan penghapusan *flow entries*.

Penelitian lain yang berjudul *A Zero Flow Entry Expiration Timeouts P4 Switch* mengimplementasikan mekanisme *flow expiry* dengan nilai *timeouts* 0. Di samping itu penelitian ini juga menambahkan mekanisme untuk mendeteksi paket FIN dan RST pada protokol TCP untuk melakukan penghapusan *flow entry* yang telah selesai melakukan komunikasi atau memutus koneksi dengan mengirimkan paket RST. Cheng-Hung menerapkan metode deteksi paket ini karena menyadari bahwa nilai *timeouts* yang besar dapat menyebabkan kondisi *flow table* penuh, sedangkan nilai *timeouts* yang kecil dapat menyebabkan beban yang berat kepada *controller*, karena harus memperbarui *flow entries* setiap saat. Dalam penelitian ini dihasilkan bahwa metode zero *timeouts* yang dikombinasikan dengan deteksi paket FIN dan RST dapat mengurangi beban *controller* dan mencegah terjadinya *flow table overflow* (He, et al., 2018).

Penelitian lainnya yang berjudul *Machine Learning Based Proactive Flow Entry Deletion for OpenFlow* menerapkan metode *flow modification* atau biasa disebut metode *proactive deletion* dengan mengirimkan pesan spesifik kepada *controller*. Dalam penelitian ini *proactive deletion* dikombinasikan dengan *machine learning* yang menghapus *flow entry* dengan mempelajari perilaku sebelumnya yang diperoleh dari statistik *flow removed message*, kemudian memprediksi *flow entry* dengan durasi paling pendek dari data sebelumnya. Selain menggunakan *machine learning*, metode *proactive deletion* ini juga diterapkan dengan algoritme FIFO dan *random deletion*. Penelitian ini berfokus pada menemukan aturan *proactive*

deletion yang paling efisien, dan hasil dari komparasi beberapa algoritme dan *machine learning* diperoleh hasil bahwa *machine learning* memiliki nilai *table missed* dan *controller overhead* paling kecil serta mampu mengatasi permasalahan *flow table overflows* (Yang & Riley, 2018).

Dari beberapa penelitian diatas yang berkaitan dengan rencana penelitian ini, diimplementasikan *hybrid flow table management* dengan menerapkan 2 mekanisme *flow removal*, yaitu *flow expiry* dan *flow modification* yang memiliki tujuan sama dengan penelitian sebelumnya yaitu manajemen *flow entries* untuk mencegah *flow table overflows*. Perbedaan dari rencana penelitian ini dengan penelitian sebelumya adalah pada penerapannya. Rencana penelitian kali ini diterapkan pada sistem *stateless server load balancing* berbasis *software defined networking*.

Sistem *server load balancing* yang diterapkan pada arsitektur SDN tentunya menjadi salah satu pemicu dari *flow table overflow*, sehingga permasalahan keterbatasan kapasitas *flow table* memungkinkan untuk diteliti lebih lanjut, terlebih jika diterapkan pada sistem *server load balancing* berbasis *software defined networking*. Pengujian dari implementasi mekanisme *flow removal* dilakukan dengan mengirimkan paket *request* dari beberapa *source* yang berbeda ke sebuah *server*. Dari serangkaian implementasi, pengujian, dan analisis diharapkan mampu mencegah *flow table overflow* dan mengetahui dampak kinerja jaringan setelah menerapkan 2 mekanisme *flow removal*, sehingga dapat mengoptimalkan sistem *server load balancing*.

2. FLOW TABLE OVERFLOW

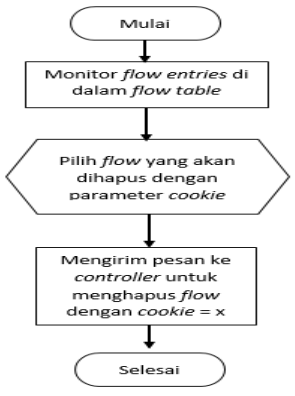
OpenFlow *switches* memiliki kapasitas *flow table* yang terbatas. Keterbatasan ruang penyimpanan ini dikarenakan *flow table* terbuat dari *Ternary Content Addressable Memory* (TCAM). TCAM memiliki konsumsi daya yang tinggi dan jejak silikon yang besar, sehingga memerlukan biaya *chip* yang mahal dan berujung kepada keterbatasan dalam menyimpan *flow entries* (Zarek, 2012).

Keterbatasan ruang penyimpanan untuk memfasilitasi *flow entries* di dalam OpenFlow *switch* dapat menyebabkan kondisi *flow table overflow*. *Flow table overflow* adalah sebuah kondisi dimana *flow table* tidak lagi dapat memfasilitasi *flow entry* baru. Ketika terjadi *flow table overflow*, maka perlu dilakukan sebuah langkah penggusuran kepada *active flow* untuk dapat memfasilitasi *flow entry* yang baru. Namun untuk meminimalisir gangguan terhadap kinerja *flow entry* dan penambahan beban *controller*, perlu dilakukan mekanisme yang tepat dalam menangani kondisi *flow table overflow* (Guo, et al., 2018).

3. METODE PENELITIAN

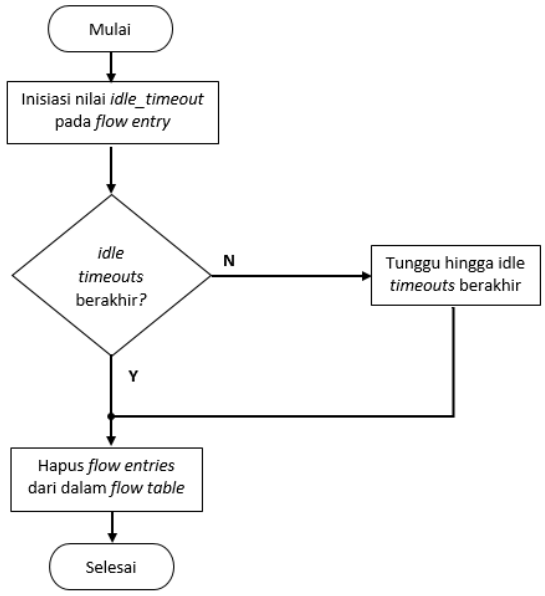
Di dalam protokol OpenFlow terdapat 2 mekanisme untuk melakukan penghapusan *flow entry* dari sebuah *flow table*. Mekanisme yang pertama adalah *flow expiry*, yang mana mekanisme ini dijalankan secara independen oleh *switch* dari *controller* berdasarkan kondisi dan konfigurasi dari *flow entry*. Setiap *flow entry* memiliki *idle timeout* dan *hard timeout* yang saling berkaitan. Jika keduanya tidak bernilai 0 maka *switch* harus mencatat waktu kedatangan dari *flow entry*, karena nantinya mungkin dibutuhkan untuk melakukan penghapusan *flow entry*.

Pemberian nilai pada *hard timeout* menyebabkan *flow entry* dihapus dari *flow table* setelah mencapai nilai atau waktu yang diberikan, tanpa peduli berapa banyak paket yang cocok. Pemberian nilai pada *idle timeout* menyebabkan *flow entry* dihapus dari *flow table* ketika tidak ada paket yang cocok selama nilai atau waktu yang diberikan. Dalam hal ini *switch* harus mengimplementasikan *flow expiry* dan menghapus *flow entry* dari *flow table* ketika salah satu nilai dari *timeout* baik *idle* atau *hard* telah terlampaui. Pada penelitian ini *flow expiry* diberikan melalui nilai *idle timeouts*. Berikut ini adalah *diagram alur* pemberian nilai *idle timeouts*.



Gambar 1. Mekanisme *Idle Timeouts*

Mekanisme kedua dalam melakukan penghapusan *flow entry* adalah *flow modification* OFPFC_DELETE atau OFPFC_DELETE_STRICT. Mekanisme ini tergolong mekanisme penghapusan yang pro aktif karena tanpa menunggu *flow expiry*, *switch* akan mengirimkan pesan kontrol eksplisit kepada *controller* untuk melakukan penghapusan (Open Networking Foundation, 2012).Berikut ini adalah *flow chart* mekanisme *flow modification.*

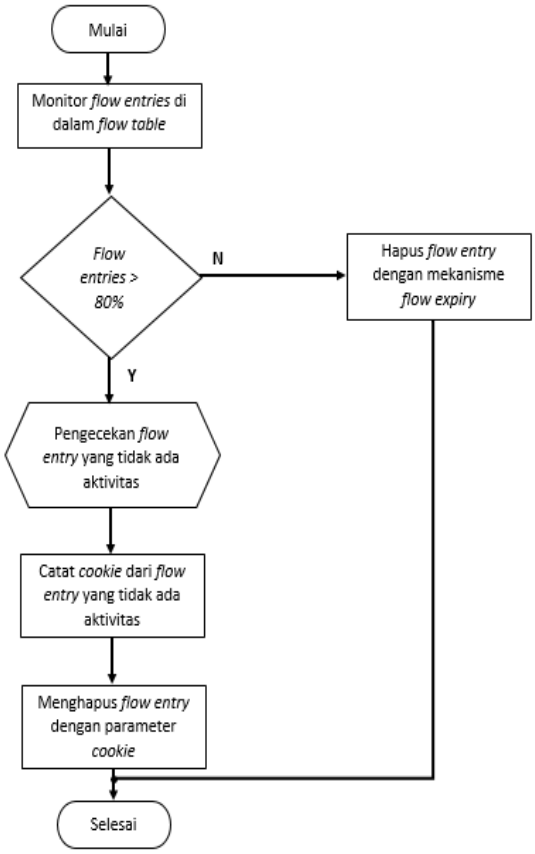


Gambar 2. Mekanisme *Flow Modification*

Penelitian ini menggunakan 2 mekanisme *flow removal*. Mekanisme pertama adalah *flow expiry* dengan pemberian nilai pada *idle timeouts.* Sedangkan mekanisme kedua adalah mekanisme *flow modification* yang biasa disebut sebagai metode *proactive deletion*. Dengan menerapkan 2 mekanisme *flow removal*, penelitian ini menerapkan *hybrid flow table management* untuk mencegah terjadinya *flow table overflow.*

Dalam penelitian ini nilai *idle timeouts* yang diberikan adalah 60 detik. Sedangkan metode *flow modification* akan mengirim pesan kontrol eksplisit kepada *controller* dengan terlebih dahulu melakukan pengecekan terhadap kondisi *flow entries*. Ketika dalam pengecekan setiap 5 detik sekali ditemukan *flow entry* yang tidak ada aktivitas, maka *switch* akan mengirimkan pesan kepada *controller* untuk menghapus *flow entry* yang tidak terdapat aktivitas dengan parameter *cookie.* Alur implementasi mekanisme *flow removal* ditunjukkan dalam Gambar 3.

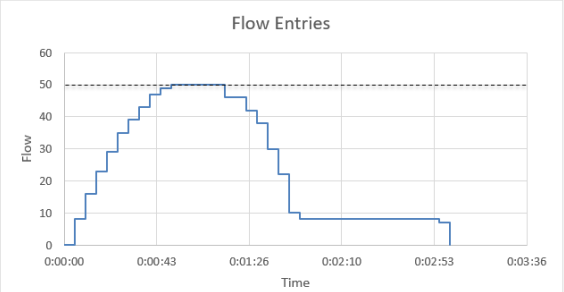
4. HASIL DAN PEMBAHASAN



Gambar 3. Mekanisme *Hybrid Flow Table Management*

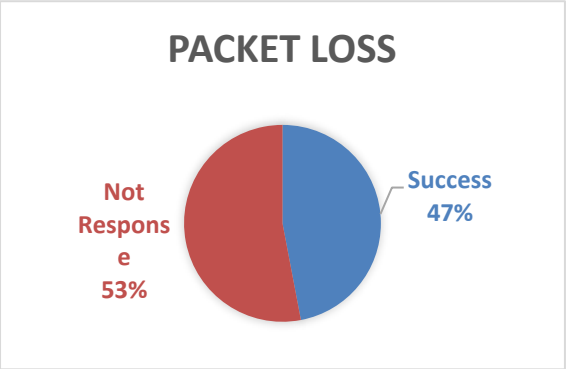
4.1. *Idle Timeouts*

Pengujian *idle timeouts* dilakukan dengan cara mengatur *idle timeouts* pada setiap *flow entry* dengan nilai 60 detik. Pengujian dilakukan dengan mengirimkan 100 paket dengan jeda tiap pengiriman paket sebesar 0.5 detik. Dalam pengujian ini, *flow entry* seharusnya dihapus dari *flow table* ketika dalam waktu 60 detik terakhir tidak ada paket yang cocok dengan *flow entry* tersebut.



Gambar 4. Grafik *Flow Entries* Hasil Pengujian *Idle Timeouts*

Berdasarkan Gambar 4, grafik tersebut menunjukkan bahwa *idle timeouts* berhasil menghapus *flow entry* setelah selama 60 detik terakhir tidak ada aktivitas. Namun tidak seluruh *request* dapat menempati *flow table* dan mendapat pelayanan dari *server*. Hal ini dapat dilihat dari grafik, yang mana *flow entries* berhenti bertambah ketika telah menyentuh angka 50, karena batas maksimal *flow table* adalah 50 *entries*. Gambar 5 menunjukkan bahwa dari 100 paket *request* yang dikirim, hanya 47% paket yang dapat dilayani oleh *server*. Sedangkan 53% lainnya tidak dapat menempati *flow table* dan tidak mendapatkan pelayanan dari *server* karena *flow table overflow*.

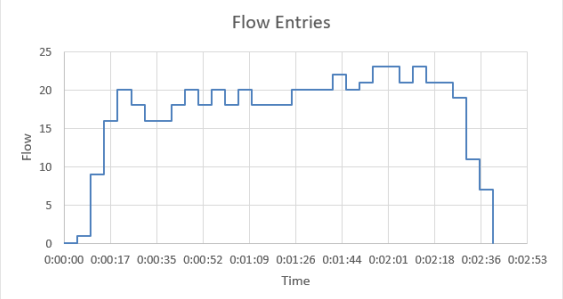


Gambar 5. Persentase *Packet Loss*

4.2. *Flow Modification*

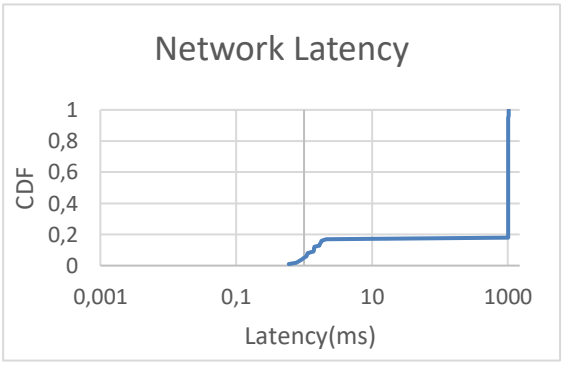
Pengujian dilakukan dengan mengirimkan 100 paket dengan jeda waktu 0.5 detik setiap kali mengirimkan paket. Dalam pengujian ini metode yang digunakan adalah *flow modification message delete flow*, dengan menggunakan *idle timeouts* 6 detik. Pada pengujian kali ini *flow entry* akan dihapus dari *flow table* ketika dalam pengecekan setiap 5 detik berikutnya tidak ada aktivitas.

Dalam pengujian ini, 100 paket *request* yang dikirimkan seluruhnya dapat dilayani oleh



Gambar 6. Grafik *Flow Entries* Hasil Pengujian *Flow Modification*

server. Gambar 6 menunjukkan bahwa kondisi *flow entries* tidak pernah mencapai batas maksimal, hanya berkisar pada angka 16 – 23 *flow entries*. Pada pengujian ini, 83% dari jumlah paket yang dikirimkan memiliki nilai *network latency* yang tinggi yaitu antara 1009 – 1035 *milisecond*. Hal ini terjadi karena ini *flow entries* langsung dihapus dari *flow table* ketika tidak ada paket yang cocok. Sehingga kondisi ini mengharuskan *controller* untuk selalu memperbarui *flow table* dan mendefinisikan *flow entry* baru setiap menerima *request* dari *source* yang baru. Beban yang diterima *controller* berdampak pada *network latency*. Hal ini dapat dilihat pada Gambar 7.

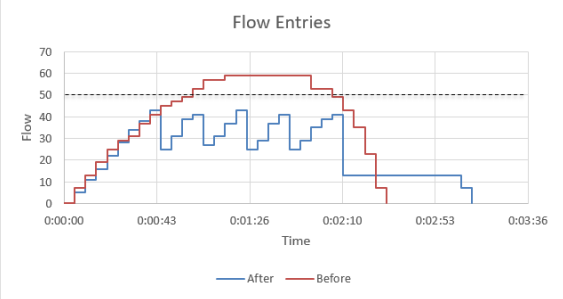


Gambar 7. Grafik *Network Latency* Hasil Pengujian *Flow Modification*

4.3. *Idle Timeouts & Flow Modification*

Pengujian *idle timeouts* dan *flow modification message* dilakukan dengan memadukan antara 2 mekanisme penghapusan *flow entries*, yaitu *flow expiry* dan *flow modification message*. Pengujian ini dilakukan dengan skenario yang telah ditetapkan sebelumnya, yaitu kapasitas *flow table* adalah 50 *flow entries* dan mengirimkan *request* sebanyak 100 kali dengan jeda pengiriman setiap 0.5 detik.

Dalam pengujian ini diberikan nilai *idle timeouts* sebesar 60 detik. *Flow modification* akan mengirimkan pesan penghapusan kepada *controller* setiap 5 detik terhadap *flow entry* yang tidak ada aktivitas. Pesan kontrol eksplisit hanya dikirimkan kepada *controller* ketika kondisi *flow table* melebihi 80% dari kapasitas maksimal.



Gambar 8. Grafik *Flow Entries* Hasil Pengujian *Idle Timeouts & Flow Modification*

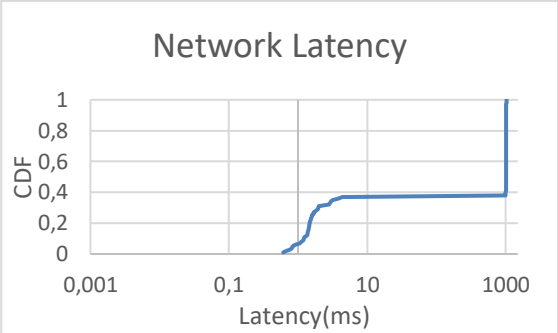
Dari gambar 8 dapat dilihat bahwa terdapat perbedaan grafik sebelum dan setelah menerapkan mekanisme *flow modification message delete flow*. Garis warna oranye menunjukkan kondisi di mana jumlah *flow entries* melebihi kapasitas maksimal *flow table* yaitu 50. Hal ini dikarenakan *flow entries* hanya dihapus dengan menggunakan mekanisme *idle timeouts* 60 detik. Garis warna biru menunjukkan kondisi *flow entries* yang stabil di bawah kapasitasi maksimal dari *flow table.*

Jumlah Paket	Sebelum	Sesudah
Success	47%	100%
Not Response	53%	0%

Gambar 9. Perbandingan Paket Loss Sebelum dan Sesudah Optimasi

Gambar 9 menunjukkan perbandingan antara *packet loss* sebelum dan sesudah menerapkan 2 mekanisme *flow removal* untuk tujuan optimasi. Sebelum menerapkan mekanisme *flow removal*, 53% dari jumlah paket yang dikirimkan tidak mendapatkan *response* dari *server* karena tidak memiliki kesempatan untuk menempati *flow table*. Sedangkan setelah menerapkan mekanisme *flow removal*, 100% dari jumlah paket *request* yang dikirimkan dapat menempati *flow table* dan mendapatkan *response* dari *server.*

Hasil performa *server* dapat dilihat dari perbedaan yang cukup signifikan antara tiap – tiap *network latency.* Hal ini dapat dilihat dengan membandingkan garis warna oranye pada Gambar 10. Berdasarkan grafik tersebut diperoleh hasil bahwa 37% *request* yang dikirimkan ke *server* memiliki nilai *latency* yang kecil yaitu antara 0,6 – 4,3 *millisecond*. Sedangkan 63% *request* yang dikirimkan ke server memiliki nilai *latency* yang cukup besar, yaitu antara 1006 – 1031 *millisecond*..



Gambar 10. Grafik *Network Latency* pada Implementasi 2 Mekanisme *Flow Removal*

Berdasarkan hasil pengujian dengan menerapkan 2 mekanisme *flow removal* dan dengan skenario yang telah ditetapkan, diperoleh hasil bahwa ketika *flow table* mendekati kapasitas maksimal, maka mekanisme *flow modification* message akan mengirimkan pesan kepada *controller* untuk melakukan penghapusan terhadap *flow entries* yang tidak ada aktivitas. Ketika kondisi *flow table* normal atau dalam hal ini kurang dari 80% dari kapasitas maksimal, disana peran *idle timeouts* berjalan. Dari hasil pengujian dapat dianalisis bahwa dengan menerapkan 2 mekanisme *flow removal* ini maka tujuan untuk mengoptimalkan *flow entries* dapat diwujudkan, terbukti dengan kapasitas *flow table* yang tidak pernah penuh sehingga semua *flow entry* baru dapat terfasilitasi dan *request* dapat dikirimkan sampai ke *server*.

5. KESIMPULAN

Berdasarkan penelitian yang telah dilakukan terhadap implementasi 2 mekanisme *flow* removal, maka dapat ditarik kesimpulan dari penelitian ini.

Pertama, setelah dilakukan pengujian dan analisis, sistem mampu mengendalikan kondisi *flow entries* agar tetap pada kondisi dibawah 80% dari kapasitas maksimal *flow table.* Dari hasil pengujian dengan mengirimkan *request* setiap 0,5 detik, diperoleh hasil bahwa sistem berhasil menangani 100% paket yang ingin menempati *flow table*, sehingga dalam hal ini tercapai tujuan optimasi *flow entries* untuk mencegah *flow table overflow* pada *round-robin server load balancing* berbasis *software defined networking.*

Kedua, berdasarkan analisis kinerja sistem *stateless server load balancing*, selain dapat melayani 100% *request* yang dikirimkan oleh *client*, diperoleh hasil bahwa penggunaan

kembali *flow entries* dapat memangkas nilai *network latency* dari komunikasi antara *client* dan *server* hingga 1031.29 *millisecond*. Dari grafik yang disediakan dalam bentuk *cumulative distribution function* (CDF, diperoleh hasil bahwa 37% *request* yang dikirimkan ke *server* memiliki nilai *network latency* yang kecil yaitu antara 0,6 – 4,3 *millisecond*. Sedangkan 63% *request* yang dikirimkan ke server memiliki nilai *network latency* yang cukup besar, yaitu antara 1006 – 1031 *millisecond*.

Beberapa saran berdasarkan hasil penelitian ini untuk pengembangan selanjutnya adalah sebagai berikut. Pertama, perlu dilakukan analisis performa dari implementasi mekanisme *flow removal* berdasarkan jenis paket yang berbeda – beda untuk memperoleh batas efisiensi sistem.

Kedua, perlu dilakukan penelitian terkait dengan menerapkan sistem pada jaringan fisik asli (OpenFlow *switch*).

DAFTAR RUJUKAN

Guo, Z. et al., 2018. Balancing Flow Table Occupancy and Link Utilization in Software Defned Networks. Future Generation Computer Systems, Volume 89, pp. Pages 213-223.

He, Cheng-Hung & Y. Chang, Brian & Chakraborty, Suchandra & Chen, Chien & Chun Wang, Li. (2018). A Zero Flow Entry Expiration Timeout P4 Switch. 1-2. 10.1145/3185467.3190785

Open Networking Foundation, 2012. *Open Networking Organization.* [Online] Available at: https://www.opennetworking.org/wp-content/uploads/.../openflow-spec-v1.3.0.pdf [Accessed 4 Januari 2019].

Yang, H. & Riley, G. F., 2018. Machine Learning Based Proactive Flow Entry Deletion for OpenFlow. 2018 IEEE International Conference on Communications (ICC), pp. 1-6.

Zarek, A., 2012. OpenFlow Timeouts Demystified.ALIF, A., 2013.