Jurnal Teknika (Jurnal Fakultas Teknik Universitas Islam Lamongan) ISSN : 2085-0859
Volume 13, No.2, Tahun 2021 e-ISSN : 2620-4770
Tersedia Online http://www.jurnalteknik.unisla.ac.id/index.php/teknika/index
DOI : 10.30736/jt.v13i2.681, Hal 99 -105

ANALISIS PERBANDINGAN *GATEWAY BALANCING* PADA *SOFTWARE DEFINED NETWORK* DENGAN ALGORITMA *SCHEDULING* TAMBAHAN

Agi Alif Ramadhan[1], Favian Dewanta[2], Ridha Muldina Negara[3]
[1,2,3]S1 Teknik Telekomunikasi, Fakultas Teknik Elektro, Telkom University
Jl. Telekomunikasi No. 1, Terusan Buahbatu, Kec. Dayeuhkolot, Bandung, Jawa Barat 40257
(022) 7566456
E-mail: agialif48@gmail.com, favian@telkomuniversity.ac.id, ridhanegara@telkomuniversity.ac.id

ABSTRACT

In the process of sending data packets on a network, of course, an architecture that has high reliability is needed. This is needed in order to guarantee the speed of delivery and the absence of data packets that fail to be sent to the destination. The availability of many lines will certainly increase availability on the network, but it does not guarantee that it will increase the speed or quality of data transmission and the success of the data getting to its destination. The existence of two or more paths allows for uneven distribution of traffic and accumulation of traffic on a gateway. To avoid this, the Gateway Balancing method will be used. In this research, gateway balancing is applied to the Software Defined Network architecture using the OpenDayLight controller by applying two scheduling algorithms, namely the Ant Colony Optimization algorithm and the Random Early Detection algorithm. By implementing gateway balancing with these two algorithms, traffic performance testing will be carried out by streaming UDP Flows. The test parameters used are Throughput, Delay, Jitter, and Packeloss. It can be concluded that traffic performance using a gateway balancing with the Ant Colony Optimization algorithm offers a better data transmission speed than the Random Early Detection algorithm with a delay difference of 4.46%. While the Random Early Detection algorithm, produces better performance on the integrity of the data, with a difference of 3.55% in throughput and 5.85% in the resulting packetloss.

Keyword**:** *Ant Colony Optimization, Gateway Balancing, OpenDayLight, Random Early Detection, SDN*

ABSTRAK

Pada proses pengiriman paket data pada suatu jaringan, tentu dibutuhkan arsitektur yang memiliki keandalan yang tinggi. Hal ini dibutuhkan agar dapat menjamin kecepatan pengiriman dan tidak adanya data yang gagal dikirimkan ke tujuan. Tersedianya banyak jalur tentu akan meningkatkan *availability* pada jaringan tersebut, namun tidak menjamin akan meningkatkan kecepatan atau kualitas pengiriman data dan keberhasilan data tersebut sampai ke tujuan. Adanya dua atau lebih jalur memungkinkan akan terjadinya penyebaran trafik yang tidak merata dan penumpukan trafik pada suatu *gateway.* Untuk menghindari hal ini, maka akan dilakukan metode *Gateway Balancing*. Pada penelitian ini dilakukan penerapan *gateway balancing* pada jaringan *Software Defined Network* menggunakan *controller* OpenDayLight dengan menerapkan dua algoritma *scheduling* yaitu algoritma *Ant Colony Optimizaion* dan algoritma *Random Early Detection.* Dengan menerapkan *gateway balancing* pada dua algoritma ini, maka akan dilakukan pengujian performansi trafik dengan mengalirkan UDP *Flows*. Parameter pengujian yang digunakan adalah *Throughput, Delay, Jitter,* dan *Packeloss.* Disimpulkan bahwa perfomansi trafik menggunakan *gateway balancing* dengan algortima *Ant Colony Optimization* menawarkan kecepatan data lebih baik dibandingkan algoritma *Random Early Detection* dengan selisih *delay* sebesar 4,46%. Sedangkan pada algoritma *Random Early Detection*, menghasilkan performa yang lebih baik pada keutuhan datanya, dengan selisih 3,55% pada *throughput* dan 5,85% pada *packetloss* yang dihasilkan.

Kata Kunci*: Ant Colony Optimization, Gateway Balancing, OpenDayLight, Random Early Detection, SDN*

1. PENDAHULUAN

Pada arsitektur jaringan konvensional(jaringan tradisional), *control plane* dan *data plane* menjadi satu dalam sebuah perangkat jaringan seperti *router.* Sedangkan pada arsitektur jaringan *Sowftware Defined Network* (SDN), *control plane* berada pada komponen yang disebut *controller* dan *data plane* tetap berada pada perangkat *switch*(Negara & Tulloh, 2017). SDN memungkinkan control jaringan dapat diprogram secara langsung yang menjadikannya dinamis, hemat biaya, dan mudah beradaptasi(*Open Networking Foundation*, n.d.). Untuk mengoptimalkan pengiriman data pada suatu jaringan, agar mendapatkan kecepatan dan keutuhan data lebih baik, maka dapat dilakukan dengan *gateway balancing. Gateway balancing* memungkinkan jaringan memerintahkan *gateway* untuk melakukan pemerataan data pada setiap *link* yang dilewati.

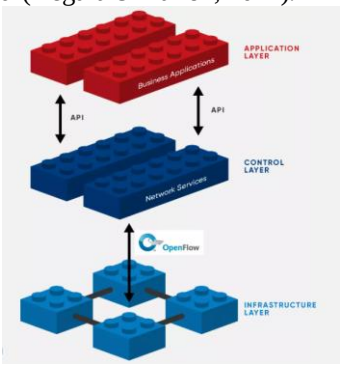
Jurnal Teknika (Jurnal Fakultas Teknik Universitas Islam Lamongan)
Volume 13, No.2, Tahun 2021
Tersedia Online http://www.jurnalteknik.unisla.ac.id/index.php/teknika/index
DOI : 10.30736/jt.v13i2.681, Hal 99 -105
ISSN : 2085-0859
e-ISSN : 2620-4770

Pada penelitian ini akan dilakukan perbandingan kinerja *gateway balancing* pada jaringan SDN menggunakan algoritma *Ant Colony Optimization* dan algoritma *Random Early Detection* dengan menggunakan *controller* OpenDayLight. Pengujian dilakukan dengan tiga skenario menggunakan peningkatan jumlah paket yang dikirimkan perdetiknya. Parameter QoS pada pengujian ini antara lain adalah *throughput, delay, jitter* , dan *packetloss.*

Adapun landasan teori yang digunakan pada penelitian ini adalah:

a. *Software Defined Network*

Dalam arsitektur jaringan SDN, *control plane* dan *data plane* yang sebelumnya satu kesatuan dipisahkan, serta infrastruktur jaringannya diabstraksi dari aplikasi. Hal ini yang membuat jaringan SDN lebih dinamis, fleksibel, dan dapat diprogram sesuai kebutuhan(*Open Networking Foundation*, n.d.). *Control plane* pada SDN terletak pada sebuah *controller* yang bertugas untuk megatur seluruh jaringan dan *data plane* tetap berada pada perangkat jaringan seperti *switch* yang bertugas meneruskan atau melakukan perintah yang diberikan oleh *controller*(Negara & Tulloh, 2017).

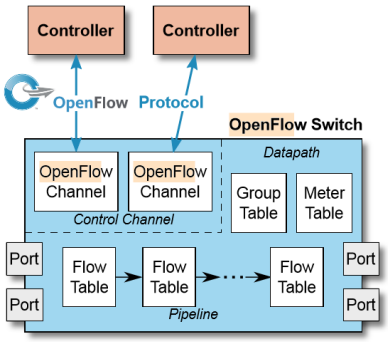


Gambar 1. Arsitektur SDN(*Open Networking Foundation*, n.d.)

Jaringan SDN menciptakan arsitektur jaringan yang mudah diprogram, karena control jaringan sudah dipisahkan dari fungsi *forwarding*-nya dan karena semua perintah terpusat pada *controller* sehingga langsung dapat menangani jaringan secara keseluruhan. Serta menjadikan jaringan lebih dinamis dan dapat beradaptasi dengan kebutuhan yang diinginkan(*Open Networking Foundation*, n.d.).

b. OpenFlow *Protocol*

Konsep OpenFlow pertama kali digagas oleh Martin Casado di Standford University pada tahun 2006 yang kemudian pada Desember 2009 sebuah versi pertama OpenFlow yaitu OpenFlow v1.0 di rilis. Sebuah organisai yang bernama Open Networking Foundation (ONF) telah mendedikasikan untuk perkembangan pada SDN telah membantu pengembangan OpenFlow dari awal hingga saat ini(*The history of OpenFlow*, n.d.)(Laboratory, 2017). OpenFlow merupakan sebuah *interface* yang berada diantara *control plane* dan *forwarding plane* pada arsitektur jaringannya.



Gambar 2. Komponen utama OpenFlow(Open Networking Foundation, 2015)

OpenFlow memungkinkan akses langsung dan dapat memodifikasi *forwarding plane* pada perangkat jaringan seperti *switch*(*Open Networking Foundation*, n.d.). OpenFlow dapat berjalan dengan satu atau lebih *flow tables* dan *group tables* untuk pencarian paket dan *OpenFlow channel* untuk berinteraksi ke *controller*. Dengan ini, *controller* dapat menambah, menghapus, dan memperbahurui *flow tables* secara reaktif maupun proaktif(*Open Networking Foundation*, 2015).

c. *Gateway Balancing*

Gateway balancing merupakan sebuah sistem yang mengatur *routing* pada suatu jaringan yang memungkinkan pemerataan trafik pada suatu *gateway* sehingga dapat mengopitmalkan *link* dan *bandwidth* yang tersedia(Hadi., 2019).

d. OpenDayLight *Controller* dan Mininet

OpenDayLight *controller* adalah sebuah *controller* untuk SDN yang dijalankan pada JVM (*Java Virtual Machine*). Sehingga dapat berjalan di berbagai sistem operasi dan perangkat keras manapun yang mendukung JAVA(*OpenDaylight*, n.d.). Mininet adalah sebuah emulator jaringan yang dapat menciptakan *host*, *switch*, *controller*¸dan *link.* Mininet menjalankan jaringan standar pada Linux dan mendukung OpenFlow sehingga dapat digunakan untuk menciptakan topologi SDN(*Mininet*, n.d.).

e. *Random Early Detection*

Random Early Detection (RED) merupakan algortima *scheduling* yang dapat mengatur perilaku trafik untuk menghindari tabrakan pada antrian. Algoritma bekerja dengan menandai sebuah paket pada antrian dengan membandingkannya dengan nilai antrian rata-

Jurnal Teknika (Jurnal Fakultas Teknik Universitas Islam Lamongan) ISSN : 2085-0859
Volume 13, No.2, Tahun 2021 e-ISSN : 2620-4770
Tersedia Online http://www.jurnalteknik.unisla.ac.id/index.php/teknika/index
DOI : 10.30736/jt.v13i2.681, Hal 99 -105

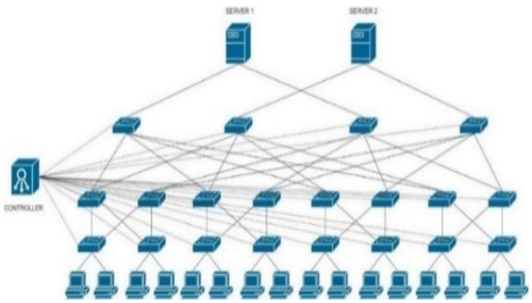
rata. Apabila nilai antrian rata-rata berada pada batas minimumnya, maka paket tidak akan ditandai, sedangkan jika berada diatas batas maksimum, maka paket akan di drop. Dan jika nilai rata-rata antrian berada diantara batas mnimum dan batas maksimum, maka akan diberikan probabilitas pada paket untuk didrop atau diteruskan(Floyd & Jacobson, 1993).

f. *Ant Colony Optimization*
Ant Colony Optimization merupakan algortima *scheduling* yang bekerja dengan mencari jalur tercepat yang tersedia pada jaringan dan dapat diperbarui. Pertama kali dikemukakan oleh Marco Dorigo pada awal tahun 1990, yang terinspirasi dari perilaku koloni semut yang mencari lintasan terpendek untuk mencari makan. Disaat koloni semut mencari makan, mereka akan mulai menjelajahi sarang mereka secara acak sambil meninggalkan jejak berupa zat feromon. Disaat semut yang dibelakangnya mencari lintasan untuk dilewati, maka semut tersebut akan memilih lintasan dengan zat feromon dengan konsentrasi yang lebih tinggi(Blum, 2005).

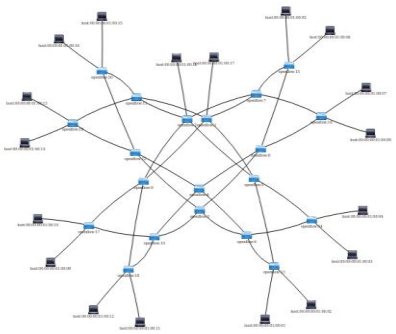
2. METODE

2.1 Desain Sistem

Sistem dijalankan pada *virtual machine* dengan sistem operasi Xubuntu 20.04 yang berjalan pada *host* Windows. Topologi yang digunakan dibangun menggunakan mininet dengan bahasa Python. Pada penelitian ini, algoritma *scheduling* untuk *gateway balancing* mengkomsumsi API dari *controller* OpenDayLight untuk membuat *logic* yang akan digunakan untuk membuat *route-flow* yang akan dilemparkan kembali ke *controller.* Pada perancangan algoritma, dilakukan pemetaan alamat seperti IP *address* dan MAC *address* untuk membuat *graph* topologi agar program dapat lebih mudah memetakan jalur yang digunakan.



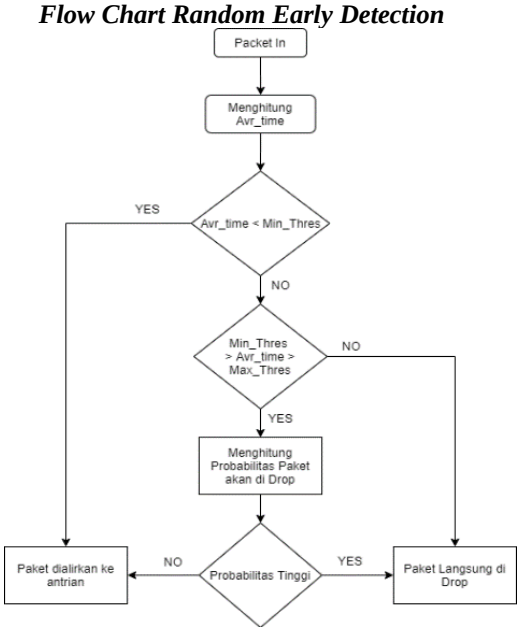
Gambar 3. Topologi *fat-tree* dengan 4 pod



Gambar 4. Simulasi topologi oleh OpenDayLight *controller*

Topolologi yang digunakan adalah topologi *fat-tree* dengan empat buah pod dengan *bandwidth link* sebesar 100Mbps. Pada topologi akan dilakukan pengujian trafik dengan *gateway balancing* menggunakan algoritma *Ant Colony Optimization* dan algoritma *Random Early Detection.* Sebuah UDP *Flows* akan dialirkan dari *host* sumber ke *host* tujuan. *Background traffic* juga akan dibangkitkan melewati beberapa jalur yang diprediksi akan digunakan oleh trafik utama untuk menambah sedikit beban pada jalurnya.

2.2 *Flow Chart Random Early Detection*



Gambar 5. Diagram alir *Random Early Detection*

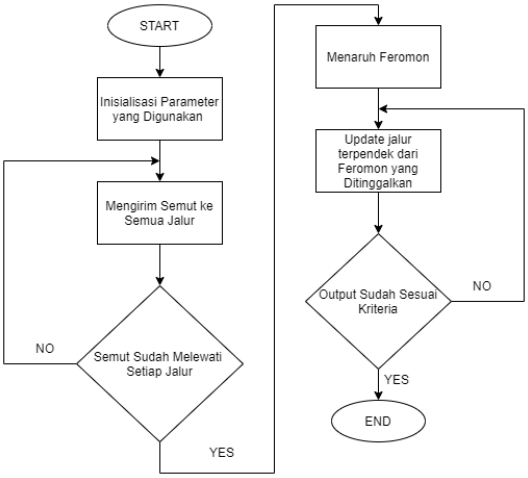
Berikut merupakan penjelasan dari *flow chart* algoritma *Random Early Detection* :

a. *Packet In*, sebuah paket masuk, dan akan dianalisis oleh program
b. Menghitung Avr_time, menghitung waktu tunggu rata-rata sebuah paket
c. Avr_time < Min_Thres, apabila waktu tunggu rata-rata berada dibawah batas minimum, maka paket akan dialirkan ke antrian

Jurnal Teknika (Jurnal Fakultas Teknik Universitas Islam Lamongan) ISSN : 2085-0859
Volume 13, No.2, Tahun 2021 e-ISSN : 2620-4770
Tersedia Online http://www.jurnalteknik.unisla.ac.id/index.php/teknika/index
DOI : 10.30736/jt.v13i2.681, Hal 99 -105

d. Min_Thres>Avr_time>Max_Thres, apabila waktu tunggu rata-ratanya berada diantara batas minimum dan batas maximum, maka akan dihitung probabilitas paket akan di *drop* atau tidak
e. Menghitung probabilitas paket akan di *drop*, melakukan penghitungan probabilitas *drop* suatu paket
f. Probabilitas tinggi, apabila probabilitas *drop* paket tinggi, maka paket akan langsung di *drop*. Dan apabila probabilitas *drop* pada paket masih rendah, maka paket akan diteruskan ke antrian

2.3 *Flow Chart Ant Colony Optimization*



Gambar 6. Diagram alir *Ant Colony Optimization*

Berikut merupakan penjelasan dari *flow chart* algoritma *Ant Colony Optimization* :

a. Start, merupakan kondisi dimana iterasi dimulai
b. Inisialisai parameter yang digunakan, menetapkan parameter yang akan digunakan pada algoritma
c. Mengirim semut ke semua jalur, memulai mengirim semut ke semua jalur/rute yang ada
d. Semut sudah melewati semua jalur, apabila semua jalur/rute yang ada sudah dilalui oleh semut, maka semut akan meninggalkan feromon pada jalur yang sudah dilaluinya. Dan apabila ada jalur yang belum dilalui oleh semut, maka algoritma akan mengulang untuk mengirimkan semut ke semua jalur/rute
e. Menaruh feromon, semut meninggalkan feromon pada jalur yang sudah di lewati sebagai parameter untuk menghitung jalur terpendek.
f. Update jalur terpendek dari feromon yang ditinggalkan, algoritma akan memperbarui informasi jalur terpendek dari feromon yang ditinggalkan
g. Output sudah sesuai kriteria, apabila output sudah sesuai dengan kriteria, maka kita mendapatkan informasi jalur terpendeknya. Apabila belum, maka algoritma akan memperbarui informasinya sesuai dengan feromon yang ditinggalkan
h. End, proses iterasi selesai dijalankan.

2.4 Skenario Pengujian

Pengujian akan dilakukan dengan tiga buah skenario, dengan memperhatikan jumlah paket yang akan dikirimkan perdetiknya dengan ukuran *default* untuk setiap paket adlah 512 bytes. Adapun jumlah paket perdetiknya yang digunakan dalam pengujian ini sebesar:

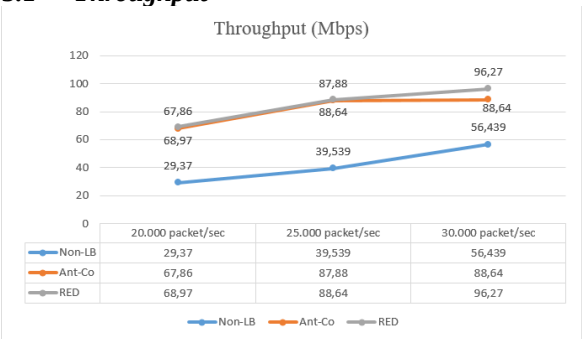
a. 20.000 *packet/s* (ekspektasi beban dibawah *bandwidth link*)
b. 25.000 *packet/s* (ekspektasi beban mendekati *bandwidth link*)
c. 30.000 *packet/s* (ekspektasi beban diatas *bandwidth link*)

Ketiga skenario diatas akan dilakukan pada tiga simulasi. Yang pertama tanpa menggunakan *gateway balancing*, lalu menggunakan *gateway balancing* dengan algoritma *Ant Colony Optimization* dan yang terakhir menggunakan *gateway balancing* dengan algoritma *Random Early Detection.* Adapun tahapan pengujian yang dilakukan pada penelitian ini, yaitu:

a. Pembangkitan *background* trafik pada jaringan dari *host* 12 sebagai *server* ke *host* 16 sebagai *client* menggunakan iperf.
b. Pembangkitan trafik utama pada jaringan dari *host* 1 sebagai *client* ke *host* 13 sebagai *server*.
c. Pembangkitan trafik utama dan pengambilan data menggunakan D-ITG.

3. PEMBAHASAN

3.1 *Throughput*



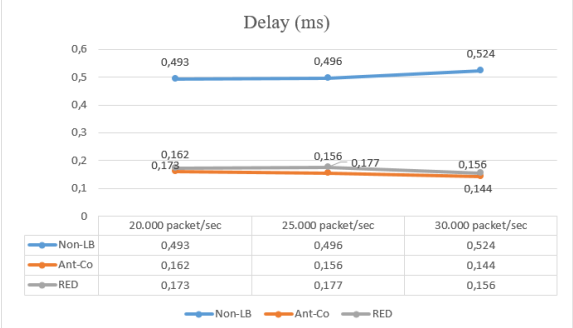
Gambar 7. Grafik *throughput*

Dapat dilihat dari grafik diatas, *throughput* yang dihasilkan saat tidak menggunakan *gateway balancing* pada *link* dengan bandwidth 100 Mbps menghasilkan *throughput* yang rendah dibandingkan saat menggunakan *gateway balancing*. Hasil yang ditunjukkan oleh algoritma *Random Early Detection* menghasilkan *throughput* sedikit lebih besar dibandingkan algoritma *Ant Colony Optimization.*

Jurnal Teknika (Jurnal Fakultas Teknik Universitas Islam Lamongan)
Volume 13, No.2, Tahun 2021
Tersedia Online http://www.jurnalteknik.unisla.ac.id/index.php/teknika/index
DOI : 10.30736/jt.v13i2.681, Hal 99 -105
ISSN : 2085-0859
e-ISSN : 2620-4770

Hal ini dikarenakan pada algoritma *Random Early Detection* memperhitungkan probabilitas paket di drop atau dialihkan berdasarkan panjang antriannya.

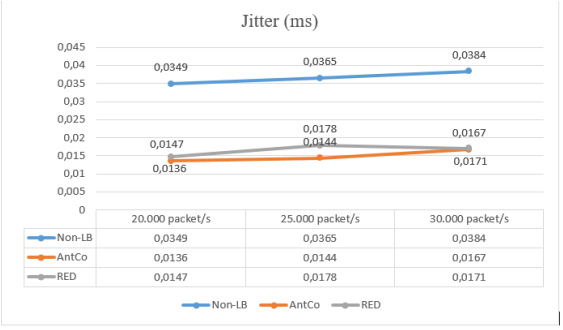
3.2 *Delay*



Gambar 8. Grafik *delay*

Dapat dilihat pada grafik diatas, *delay* pada saat sebelum menggunakan *gateway balancing* cukup besar dan semakin banyak paket yang dikirimkan perdetik maka besar *delay*-nya pun semakin besar. Berbeda pada saat dilakukan *gateway balancing*, *delay* yang dihasilkan lebih kecil dikarenakan paket dialirkan ke beberapa jalur oleh *gateway* dan menggunakan jalur terpendeknya. Disini algoritma *Ant Colony Optimization* unggul dengan besar *delay* yang tidak terlalu besar dari algoritma *Random Early Detection* hanya sebesar 0,1%.

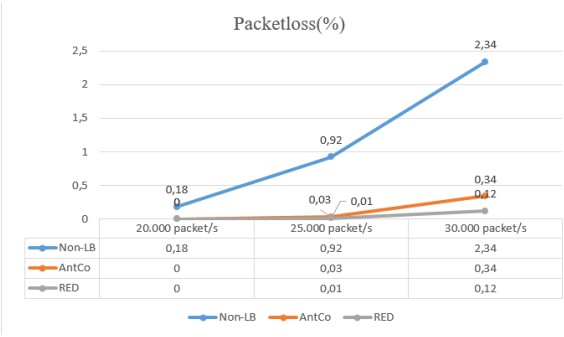
3.3 *Jitter*



Gambar 9. Grafik *jitter*

Dapat dilihat pada grafik diatas, *jitter* yang dihasilkan oleh trafik tanpa menggunakan *gateway balancing* lebih besar daripada saat sesudah menggunakan *gateway balancing.* Disini *jitter* pada algoritma *Random Early Detection* mengalami peningkatan performa yang dapat dilihat pada grafiknya yang turun. Sedangkan pada algoritma *Ant Colony Optimization* mengalami peningkatan seiring bertambah banyaknya jumlah paket yang dikirimkan.

3.4 *Packetloss*

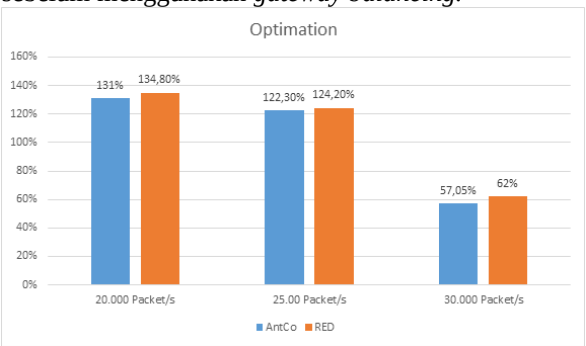


Gambar 10. Grafik *packetloss*

Dapat dilihat pada grafik diatas, *packetloss* yang dihasilkan sebelum menggunakan algoritma *gateway balancing* pada trafik meningkat dengan signifikan seiring bertambah banyaknya paket yang dikirimkan. Sedangkan setelah menggunakan algoritma *gateway balancing* pada trafik, *paacketloss* yang dihasilkan sangat kecil. Hal ini terjadi karena paket yang dikirimkan dialirkan tidak hanya melalui satu jalur dan menggunakan jalur terbaiknya. Pada skenario pengiriman 30.000 paket/s, algoritma *Random Early Detection* mendapatkan *packetloss* yang lebih kecil dari algoritma *Ant Colony Optimization.* Hal ini dimungkinkan terjadi karena pada algoritma *Random Early Detection* mengoptimalkan antrian paket pada trafiknya.

3.5 Hasil Optimasi

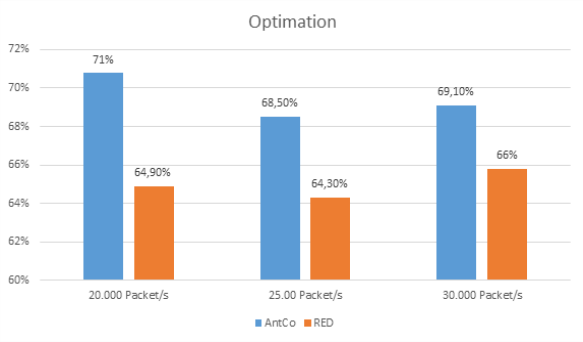
Hasil optimasi merupakan besar peningkatan performa trafik dengan menghitung selisih dari hasil parameter QoS yang diamati pada trafik setelah menggunakan *gateway balancing* dengan trafik sebelum menggunakan *gateway balancing.*



Gambar 11. Optimasi *throughput*

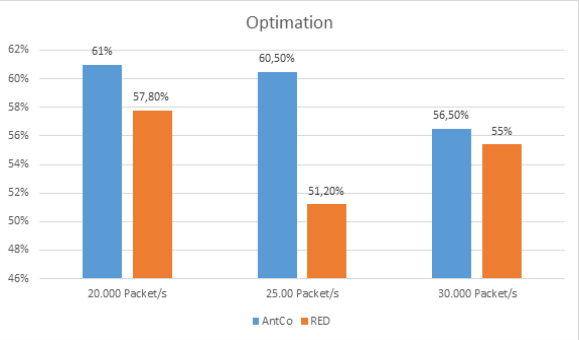
Berdasarkan *chart* diatas, trafik yang menggunakan *gateway balancing* dengan algoritma *scheduling Random Early Detection* mengalami optimasi pada *throughput* lebih besar dibandingkan dengan menggunakan algoritma *scheduling Ant Colony Optimization* dengan selisih rata-rata sebesar 3,55 %.

Jurnal Teknika (Jurnal Fakultas Teknik Universitas Islam Lamongan)
Volume 13, No.2, Tahun 2021
Tersedia Online http://www.jurnalteknik.unisla.ac.id/index.php/teknika/index
DOI : 10.30736/jt.v13i2.681, Hal 99 -105
ISSN : 2085-0859
e-ISSN : 2620-4770



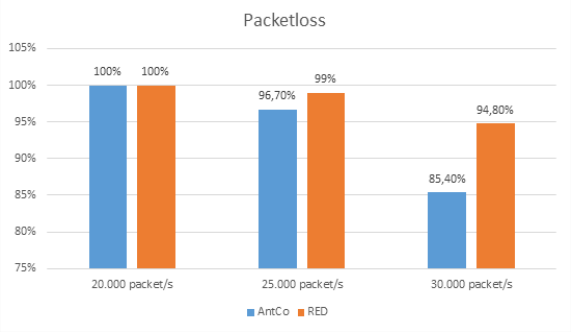
Gambar 12. Optimasi *delay*

Berdasarkan *chart* diatas, trafik yang menggunakan *gateway balancing* dengan algoritma *scheduling Ant Colony Optimization* mengalami optimasi pada *delay* lebih besar dibandingkan dengan menggunakan algoritma *scheduling Random Early Detection* dengan selisih rata-rata sebesar 4,46 %.



Gambar 13. Optimasi *jitter*

Berdasarkan *chart* diatas, trafik yang menggunakan *gateway balancing* dengan algoritma *scheduling Ant Colony Optimization* mengalami optimasi pada *jitter* lebih besar dibandingkan dengan menggunakan algoritma *scheduling Random Early Detection* dengan selisih rata-rata sebesar 4,6 %.



Gambar 14. Optimasi *packetloss*

Berdasarkan *chart* diatas, trafik yang menggunakan *gateway balancing* dengan algoritma *scheduling Random Early Detection* mengalami optimasi pada *packetloss* lebih besar dibandingkan dengan menggunakan algoritma *scheduling Ant Colony Optimization* dengan selisih rata-rata sebesar 5,85 %.

4. KESIMPULAN

Dari hasil pengujian dengan tiga skenario diatas dapat disimpulkan bahwa:

1. *Delay* yang dihasilkan pada trafik yang sudah menggunakan *gateway balancing* mendapatkan hasil yang lebih kecil daripada sebelum trafik menggunakan *gateway balancing.* Disini algoritma *Ant Colony Optimizatiuon* menghasilkan optimasi *delay* yang lebih baik dengan rata-rata selisih sebesar 4,46% dari algoritma *Random Early Detection.* Hal ini dapat terjadi karena algoritma *Ant Colony Optimization* menggunakan jalur terpendek yang tersedia dan dapat diperbaharui.
2. *Troughput* yang dihasilkan pada trafik setelah menggunakan *gateway balancing* mendapatkan hasil yang lebih baik dan cukup signifikan setelah menggunakan *gateway balancing.* Disini algoritma *Random Early Detection* mendapatkan hasil optimasi yang lebih baik dengan selisih sebesar 3,35% dibandingkan dengan algoritma *Ant Colony Optimization* dikarenakan pada algoritma ini mengoptimalkan antrian pada trafiknya. Dapat dilihat pada grafik *Troghoput* sebelumnya, semakin besar jumlah paket yang dikirimkan per detiknya, pada algoritma *Ant Colony Optimiztaion* mengalami penurunan dikarenakan tidak melakukan pengoptimalan antrian pada trafiknya.
3. *Jitter* yang dihasilkan pada trafik setelah menggunakan *gateway balancing* mengalami peningkatan performa dibandingkan dengan sebelum menggunakan *gateway balancing.* Dimana algoritma *Ant Colony Optimization* unggul dengan rata-rata selih sebesar 4,6% dari algoritma *Random Early Detection*. Ini menandakan bahwa dari dua algioritma ini sudah menghasilkan trafik yang cukup stabil.
4. *Packtloss* yang dihasilkan pada trafik setelah menggunakan *gateway balancing* lebih kecil dibandingkan dengan sebelum menggunakan *gateway balancing.* Pada algoritma *Random Early Detection* mendapatkan hasil optimasi yang lebih baik dengan selisih sebesar 5,85% dibandingkan dengan algorimta *Ant Colony Optimization.* Hal ini bisa terjadi dikarenakan pada algoritma *Random Early Detection* melakukan pengoptimalan antrian pada trafiknya.

PUSTAKA

Blum, C. (2005). Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews*, *2*(4), 353–373. https://doi.org/10.1016/j.plrev.2005.10.001

Floyd, S., & Jacobson, V. (1993). Random Early

ISSN : 2085-0859
e-ISSN : 2620-4770

Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, *1*(4), 397–413. https://doi.org/10.1109/90.251892

Hadi, F. S., Sofia Naning Hertiana S.T., M. ., & Ridha Muldina Negara S.T., M. T. (2019). *Analisis Performansi Gateway Balancing Pada Wireless Software Defined Network (WSDN)*.

Laboratory, C. N. (2017). Introduction to OpenFlow Division : Software Defined Network Sejarah OpenFlow. *Modul Software Defined Network*.

Mininet Overview - Mininet. (n.d.). Retrieved July 27, 2021, from http://mininet.org/overview/

Negara, R. M., & Tulloh, R. (2017). Analisis Simulasi Penerapan Algoritma OSPF Menggunakan RouteFlow pada Jaringan Software Defined Network (SDN). *Jurnal Infotel*, *9*(1), 75. https://doi.org/10.20895/infotel.v9i1.172

Open Networking Foundation. (2015). OpenFlow Switch Specification (Version 1.5.1). *Current*, *0*, 1–36. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf

OpenDaylight Controller Overview — OpenDaylight Documentation Silicon documentation. (n.d.). Retrieved July 27, 2021, from https://docs.opendaylight.org/en/stable-silicon/user-guide/opendaylight-controller-overview.html

Software-Defined Networking: The New Norm for Networks - Open Networking Foundation. (n.d.). Retrieved July 27, 2021, from https://opennetworking.org/sdn-resources/whitepapers/software-defined-networking-the-new-norm-for-networks/

Software-Defined Networking (SDN) Definition - Open Networking Foundation. (n.d.). Retrieved July 27, 2021, from https://opennetworking.org/sdn-definition/

The history of OpenFlow. (n.d.). Retrieved July 27, 2021, from https://www.computerweekly.com/feature/The-history-of-OpenFlow