ISSN 2355-3286

Rancang Bangun Mekanisme *Load Sharing* Pada *Link Aggregation* Menggunakan *Software Defined Networking*

Samuel[1], Cahyo Eko Samudera[2]
Computer Engineering, Universitas Multimedia Nusantara, Tangerang, Indonesia
samuel.hutagalung@umn.ac.id[1], cahyosamudera@gmail.com[2]

Diterima 14 Mei 2017
Disetujui 16 Juni 2017

Abstract— ***Given the level of activity of Internet users is high on lately. Impact on the growing needs for server throughput can be treated with load sharing on link aggregation. However, it possess a problem for UDP protocol data packets which will experience jitter and packet received out of order high. Therefore, the author tries to create an algorithm to share the package which will be sent to the server with weighted round robin method that focuses UDP Protocol data packets. The author uses the Software Defined Networking (SDN) and OpenFlow protocol that is capable of directly reprogrammed network devices. Host connect by sending packets Internet Control Message Protocol (ICMP), Transport Control Protocol (TCP) and User Datagram Protocol (UDP) to the server, then do an analysis of the bandwidth, jitter, datagrams, and retry. The author has successfully implemented with the emulator Mininet and testing. The results indicate that the average jitter is able to be reduced to 50% and packets received out of order is reduced to 0 compared to standard link aggregation with weighted round robin load sharing method.***

Keywords— ***Software Defined Netwoking, Link Aggregation, Weighted Round Robin, UDP.***

I. PENDAHULUAN

Perangkat jaringan komputer, seperti *Router* dan *Switch* memiliki *Management Interface*, sehingga dapat dikonfigurasikan dan dikelola oleh operator. *Command Line Interface (CLI)* dan *Graphical User Interface (GUI)* merupakan contoh dari *Management Interface* [1]. *Management Interface* merupakan salah satu fungsi dari Sistem Operasi yang terdapat pada perangkat jaringan komputer. Sistem Operasi dan fungsi-fungsi di dalamnya bersifat *embedded*. Dengan demikian, tidak memungkinkan apabila seseorang ingin mengubah atau mengoptimalkan cara kerja dari suatu fungsi dalam perangkat jaringan komputer.

Dengan melakukan pemisahan terhadap *data-plane* dan *control-plane* pada perangkat jaringan komputer, Sistem Operasi dan cara kerja dari fungsi-fungsi di dalamnya dapat diubah sesuai dengan yang diinginkan. Pemisahan inilah yang mendasari terbentuknya paradigma baru dalam jaringan komputer yang disebut dengan *Software Defined Networking (SDN)* [2].

Dengan cara ini, pengguna *SDN* dapat membuat sendiri sebuah fungsi pada perangkat jaringan komputer yang akan dijalankan pada *Platform Controller*. *Platform Controller* juga memiliki *Application Programming Interfaces (APIs)* untuk mempermudah pengguna dalam mengimplementasikan fungsi yang dibuat pada perangkat jaringan komputer atau mengubah cara kerja dari fungsi yang sudah ada sebelumnya. Dengan menggunakan *SDN*, diharapkan ketergantungan terhadap *vendor* tertentu bisa diminimalisir, memungkinkan adanya fleksibilitas dalam hal modifikasi fungsi pada perangkat jaringan komputer sesuai dengan kebutuhan [16].

Melihat tingkat aktivitas dari pengguna internet yang tinggi, tentu saja hal ini akan berdampak pada penyedia informasi. Kinerja *web* & *database server* sebagai penyedia layanan konten selalu diharapkan dapat memenuhi semua kebutuhan dari pengguna. Penerapan *Link Aggregation* sangat diperlukan untuk meningkatkan *throughput* dan menyediakan *fault tolerance* apabila ada *Link* yang bermasalah. *Load Sharing* pada *Link Aggregation* merupakan teknik yang berfungsi untuk membagi beban *traffic* kepada beberapa *Link* dengan tujuan agar *Link* tidak mengalami *overload* [3].

Salah satu metode *Load Sharing* yang populer digunakan adalah *Round Robin*. Metode *Round Robin* ini merupakan metode *Load Sharing* yang bersifat statis dan selalu *balance*. Metode *Round Robin* memiliki algoritma penjadwalan yang kurang efektif untuk diterapkan pada dunia nyata, karena setiap *Link* akan memiliki penggunaan yang berbeda. Metode lain yang mampu menutupi kelemahan ini adalah *Weighted Round Robin*. Algoritma penjadwalan pada *Weighted Round Robin* hampir sama dengan *Round Robin*, perbedaannya metode ini dapat memberikan prioritas terhadap Link yang digunakan.

Beberapa contoh jenis paket data yang dapat dikirimkan lewat *Link Aggregation* dengan cara *Load Sharing* berupa *Internet Control Message Protocol (ICMP) Packet, Transport Control Protocol (TCP)*

based Packet dan *User Datagram Protocol (UDP) based Packet*. Akan tetapi terdapat masalah pada paket yang menggunakan protokol *UDP* ketika dikirimkan dengan cara *Load Sharing*. Hal ini disebabkan karena protokol *UDP* tidak memiliki *sequence number* sehingga paket yang menggunakan protokol *UDP* tidak dapat disusun kembali sesuai dengan urutan pengiriman apabila sampai di tujuan dengan urutan acak. Protokol *UDP* umumnya digunakan oleh aplikasi ringan yang membutuhkan kecepatan komunikasi dan tidak transaksional, seperti *Domain Name System (DNS)* dan *Real-Time Transport Protocol (RTP)*. Sebagaimana diketahui bahwa *DNS* digunakan untuk menerjemahkan nama *domain* menjadi alamat IP, sedangkan *RTP* digunakan untuk mengirimkan *audio* dan *video* melalui jaringan berbasis IP.

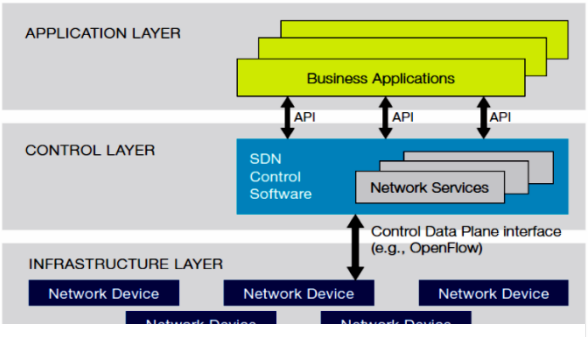
Dengan latar belakang dan permasalahan tersebut, penulis melakukan penelitian untuk merancang mekanisme *Load Sharing* pada *Link Aggregation* dengan metode *Weighted Round Robin* yang memperhatikan *UDP based Packet* pada *Software Defined Networking*.

II. TINJAUAN PUSTAKA

Di dalam bab ini, penulis mengambil beberapa tinjauan pustaka yang dijadikan sebagai landasan pengerjaan aplikasi *Link Aggregation* dengan *Load Sharing* bermetode *Weighted Round Robin* pada *Software Defined Networking (SDN)* dengan menggunakan *Ryu Controller*.

A. *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 [4]. Pemisahan *data-plane* dan *control-plane* ini memungkinkan untuk memprogram perangkat tersebut sesuai dengan yang diinginkan secara terpusat (*SDN Controller*), sehingga hal ini memungkinkan untuk mengontrol, memonitor, dan mengatur sebuah jaringan komputer dari sebuah titik (*node*) terpusat tersebut.



Gambar 1. Ilustrasi dari Arsitektur SDN [2]

Melalui *OpenFlow*, kita tidak hanya dapat melakukan *flow forwarding* berbasis *network layer* tetapi juga dapat dilakukan pengaturan pergerakan paket data secara terpusat mulai dari *layer* 2 sampai *layer* 7 *forwarding* (*flow granuality*), sehingga aliran paket data di jaringan dapat diprogram secara independen [5]. Hal ini dapat dilakukan dengan membuat algoritma dan *forwarding rules*-nya pada *controller* kemudian aturan tersebut didistribusikan ke *switch* yang ada di jaringan. Terdapat beberapa *OpenFlow Controller* yang dapat digunakan seperti *NOX (C base)*, *Ryu (phyton base)*, dan *Floodlight (java base)*.

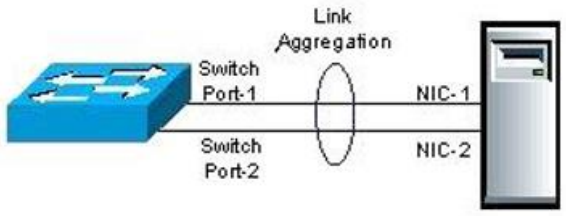
Proses aliran paket data pada *Switch Openflow* [6] yang pertama, Ketika sebuah paket data tiba di *switch OpenFlow*, bagian *header* paket diperiksa berdasarkan entri *flow table*. Lalu, jika entri yang sesuai ditemukan, *Switch* kemudian menerapkan instruksi-instruksi terkait berdasarkan aliran paket data yang sesuai juga. Terakhir, jika tidak ada yang sesuai dengan prosedur *flow table*, maka aliran paket data akan diarahkan ke entri *table-miss*. Entri *table-miss* adalah entri yang diperlukan yang menentukan set instruksi yang akan diterapkan terhadap paket data yang masuk ketika tidak ada yang cocok atau sesuai dengan prosedur *flow table*. Instruksi mencakup: Menghapus (*dropping*) paket; Mengirim paket pada semua *interface*; dan Meneruskan paket ke *controller*.

Gambar 2. *Flowchart* Proses Pengiriman Data [6]

B. *Link Aggregation*

Link Aggregation (LAg) adalah bahasa standar teknologi jaringan komputer yang menggabungkan beberapa *link*/*trunk*/*cable*/*port* fisik secara paralel untuk mendapatkan kapasitas *Bandwidth* yang lebih besar dan untuk memberikan *redundancy* dengan menggunakan teknologi *Ethernet*. Beberapa *vendor* memiliki standar masing-masing untuk penerapan *LAg*, namun standar internasional yang umum digunakan

untuk *LAg* adalah *IEEE802.3ad*. Istilah lain *LAg* ini adalah *Link Aggregation Control Protocol (LACP).*



Gambar 3. *Link Aggregation* pada *Single Server* [11]

Penggabungan beberapa *link* menggunakan *LAg* harus dari *link* yang memiliki kapasitas yang sama, seperti sesama 10Mb/s atau sesama 100Mb/s. Kapasitas yang lain yang memungkinkan untuk digabungkan adalah 1Gb/s dan 10Gb/s.

Kombinasi dari berbagai kapasitas tidak dimungkinkan. Peningkatan kapasitas dalam *LAg* sesuai dengan jumlah *link* yang digunakan terhadap kapasitas individu *link*. Misal, penggabungan tiga *link* 10Mb/s akan mendapatkan 30Mb/s. Begitu juga dengan penggabungan lima *link* 100Mb/s akan mendapatkan kapasitas 500Mb/s. Proteksi *link* menggunakan *LAg* tidak dimaksudkan untuk menambah kapasitas tetapi adalah untuk *high availability.*

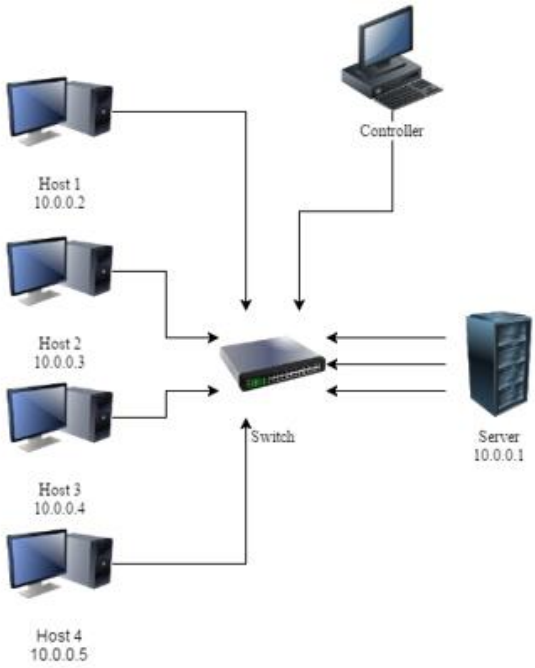
C. Weighted Round Robin

Penjadwalan *Weighted Round Robin* merupakan evolusi dari penjadwalan *Round Robin*. Setiap *Link* dapat diberi bobot berupa *integer* yang mengindikasikan kapasitas performanya. *Link* yang mempunyai bobot lebih tinggi akan mendapatkan jatah terlebih dahulu dari pada *Link* yang mempunyai bobot lebih rendah dan *Link* yang mempunyai bobot lebih besar akan mendapatkan mendapatkan jatah lebih banyak dibandingkan dengan *Link* yang mempunyai bobot lebih kecil [13].

Satu siklus WRR dapat diketahui dari total bobot prioritasnya. Contoh, terdapat tiga *Link* yaitu A, B, dan C yang masing-masing mempunyai bobot 4, 3, dan 2. Maka, 1 siklus WRR adalah 9 dari penjumlahan 4 + 3 + 2 = 9. Proses distribusi pembagian jatah dimulai dari prioritas tertinggi ke terendah dengan proses giliran *Round Robin*. Penulis menggunakan teknik *smooth service* yang terkenal lebih baik [13]. Oleh karena itu, urutan penjadwalan sesuai dengan *smooth service* untuk kasus ini adalah AABABCABC untuk setiap 1 siklus. Apabila prioritas diatur menjadi sama maka akan menjadi *balanced*. Singkatnya, penjadwalan *Round Robin* adalah penjadwalan *Weighted Round Robin* yang menganggap semua bobot *Link* sama.

III. PERANCANGAN SISTEM

Pada bab ini akan dijelaskan tentang proses perancangan sistem yang digunakan dalam penelitian.



Gambar 4. Rancangan Topologi Sistem

A. Analisis Pemilihan Link

Topologi yang akan dibuat memiliki jumlah *Link* ditentukan sebanyak empat unit *Link*, satu unit *Server*, satu unit *Switch*, dan satu unit *Controller*. Semua perangkat ini akan disimulasikan dalam *emulator Mininet*. Selanjutnya, analisa dilakukan terhadap paket data dengan protokol *TCP* dan *UDP* pada *Link Aggregation* dengan metode *Load Sharing Weighted Round Robin*.

Pada proses pemilihan Link, paket data yang masuk dianalisis dan dikategorikan menjadi *UDP based Packet* atau *non-UDP based Packet*. Berdasarkan topologi pada Gambar 4, ada satu unit Server yang terhubung dengan satu unit Switch yang dikendalikan oleh satu unit Controller. Semua perangkat tersebut akan disimulasikan dalam emulator Mininet.

Setiap data yang masuk ke *Switch* akan di-*parse* untuk bisa didapatkan detil yang lebih jauh. Pada saat proses *parsing* setiap data akan diperiksa tipe protokol, alamat *IP*, dan *MAC* tujuan dan sumbernya. Layaknya tukang pos, setelah diketahui tujuan dan sumbernya, apabila tujuan sebuah paket tersebut tidak menuju ke *Server*, maka paket akan dikirim langsung ke *port* tujuan sesuai dengan *CAM table* yang telah dipelajari oleh *Switch*. Sedangkan apabila tujuan dari suatu paket ke *Server* maka paket harus diproses lebih lanjut.

Semua paket yang menuju *Server* akan diperiksa, apakah menggunakan protokol *UDP* atau tidak. Apabila protokol tersebut menggunakan *UDP* maka sebelum dilakukan *Load Sharing* akan diperiksa sumber IP dan *port* dari paket tersebut. Jika paket data tersebut menggunakan *UDP* dan alamat sumber IP dan *port*nya belum pernah dikenali oleh *Switch* maka

alamat sumber IP tersebut akan disimpan di *Link* bersangkutan setelah proses *Load Sharing*. Sedangkan apabila paket data tersebut berprotokol *UDP* dan alamat sumber IP dan *port*nya telah dikenali sebelumnya. Maka paket data tersebut langsung dikirimkan ke *Link* yang sama dengan alamat IP dan *port* sumber yang sebelumnya tanpa harus melalui proses *Load Sharing*.

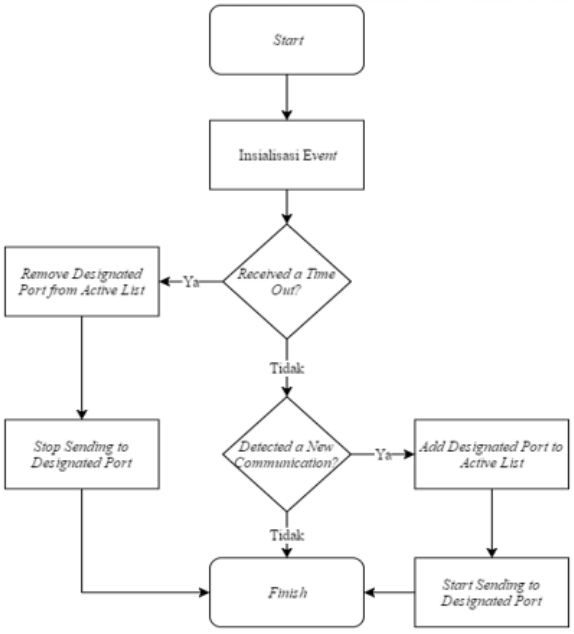
Tetapi jika dari awal paket data tersebut memang bertujuan ke *Switch* sedangkan memiliki tipe protokol yang *non-UDP* maka setiap paket data tersebut harus melalui proses *Load Sharing* sebelum dikirim ke *Link* tujuan.

Tabel 1. Alamat IP dan MAC *Host*

Nama *Host*	**Alamat *IP***	**Alamat *MAC***
H1	10.0.0.2	00:00:00:00:00:22
H2	10.0.0.3	00:00:00:00:00:23
H3	10.0.0.4	00:00:00:00:00:24
H4	10.0.0.5	00:00:00:00:00:25

B. Event Handling

Jika pada situasi ketika suatu *Link* aktif atau mati secara tiba-tiba yang sedang dalam kondisi *Link Aggregation*, maka *Ryu Controller* bisa mengirim suatu *event* berupa "*EventSlaveStateChanged*". *Event* tersebut akan berisi bagaimana *Ryu Controller* mengatasi masalah tersebut pada kondisi *Link Aggregation* untuk menjamin kelangsungan komunikasi dan meningkatkan *fault tolerance*. Oleh karena itu, Dengan Gambar 5 di bawah ini, penulis memaparkan aplikasi untuk *Ryu Controller*.



Gambar 5. *Flowchart Event Handler*

Event bisa terpicu ketika terjadi *event* yang bisa berupa *request time out* atau pun terdeteksi *Link* baru. Pada suatu ketika ada *Link* yang secara tiba-tiba berhenti berkomunikasi dan melewati batas waktu maka secara otomatis *Switch* akan menganggap telah terjadi *event time out* yang memicu *Ryu Controller* mengirim suatu *event* berupa "*EventSlaveStateChanged*". Lalu *Ryu Controller* menghapus *port* yang dianggap mati tersebut dari *active list* kemudian memerintahkan *Switch* untuk tidak mengirim apapun melalui *port* tersebut. Apabila suatu ketika ada *Link* yang tiba-tiba berkomunikasi kembali, *Switch* akan memberitahu *Controller* bahwa ada *Link* yang mencoba berkomunikasi. *Ryu Controller* akan segera menambahkan *port* dimana *Link* yang sedang mencoba berkomunikasi itu ke *active list* kemudian memerintahkan *Switch* untuk bisa mengirim paket data melalui *port* tersebut.

IV. Implementasi dan Pengujian

Dalam proses implementasi ini, hal yang pertama dilakukan ialah masing-masing *Host* akan melakukan *ping* pada alamat *IP* 10.0.0.1, yang merupakan alamat *Server* untuk memastikan semua *Host* bisa berkomunikasi dengan *Server*. Selanjutnya, pengiriman paket data *TCP* dan *UDP* yang diikuti dengan pengujian *fault tolerance* dan analisis data.

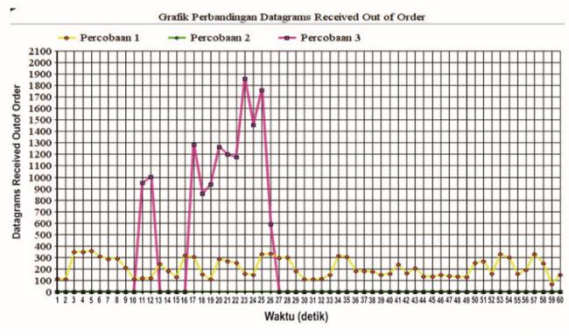
A. *Pengujian Terhadap Paket User Datagram Protocol*

Proses pengujian terhadap paket data UDP akan dilakukan dengan tool Iperf dari Host ke Server. Host akan mengirim paket UDP selama 60 detik melalui port tujuan 3333 dengan bandwidth 10Mbps. Informasi tentang percobaan UDP dipaparkan pada Tabel 2 di bawah ini.

Tabel 2. Tabel Percobaan UDP

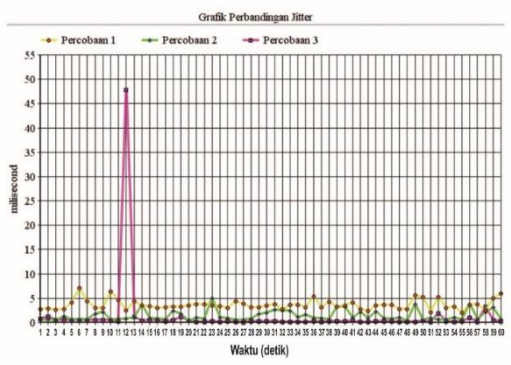
No.	**Percobaan**
1	Metode *Load Sharing* dengan *Weighted Round Robin* tanpa menggunakan algoritma yang penulis buat.
2	Metode *Load Sharing* dengan *Weighted Round Robin* dengan menggunakan algoritma yang penulis buat.
3	Sama seperti nomor kedua namun ditambah apabila terjadi ada *Link* yang mati pada detik ke-5 dan aktif pada detik ke-16.

Hasil dari pengujian tersebut ditampilkan dengan grafik pada gambar 6, 7, dan 8.



Gambar 6. Grafik Perbandingan *Datagrams Received Out of Order*

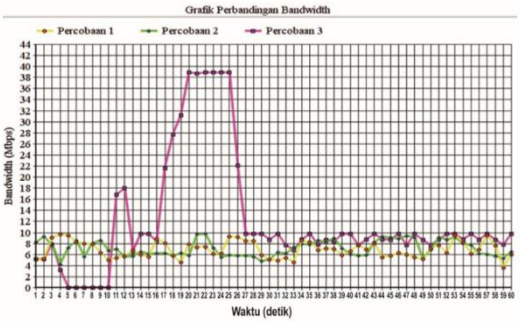
Dari Gambar 6 di muka terlihat bahwa percobaan pertama menghasilkan semua pengiriman paket UDP sampai ke Server selalu out of order. Pada percobaan kedua semua paket UDP yang sampai ke Server tidak ada yang mengalami out of order. Pada percobaan terakhir, paket UDP mulai mengalami out of order ketika di detik ke-5. Link pertama yang menjadi jalur awal untuk mengirim paket UDP mengalami masalah. Di detik ke-13, *Switch* telah selesai *learning* dan menggunakan *Link* lainnya untuk mengirim paket UDP sehingga tidak ada paket yang mengalami *out of order*. Kemudian di detik ke-16, *Link* pertama terdeteksi kembali aktif yang menyebabkan *Switch* kembali *learning* untuk memilih jalur untuk pengiriman paket *UDP* hingga detik ke-27. Mulai detik ke-28, *Switch* telah selesai *learning* sehingga tidak ada paket yang mengalami *out of order* seperti semula.



Gambar 7. Grafik Perbandingan *Jitter*

Dari Gambar 7 terlihat bahwa *Jitter* pada percobaan pertama rata-rata lebih tinggi hingga 100% dari percobaan kedua. *Jitter* adalah variasi jeda dalam *miliseconds* antar paket data yang diterima. Pada jaringan yang sehat semakin kecil nilai *Jitter* semakin bagus. Semakin tinggi nilai *Jitter* bisa menyebabkan *packet loss* dan mengurangi *quality of service* untuk protokol yang sensitif terhadap waktu seperti *VoIP*. Pada percobaan ketiga, di detik ke-11 terlihat ada peningkatan *Jitter* yang tinggi karena ada *Link* 1 terdeteksi mati pada detik ke-5. Namun, itu tidak lama karena *Switch* secara cepat beradaptasi pada situasi. Pada detik ke-16 dimana ada *event Link* 1 terdeteksi kembali aktif, tidak nampak perubahan *Jitter* yang spesifik karena *Switch* secara otomatis beradaptasi.

Dari Gambar 8 terlihat bahwa perbedaan *Bandwidth* pada percobaan pertama dan kedua tidak memiliki perbedaan yang signifikan. Akan tetapi pada percobaan ketiga terlihat ada gejolak perubahan *Bandwidth* hingga 400%. Karena pada detik ke-5, *Link* 1 mati yang menyebabkan tidak ada paket *UDP* yang diterima oleh *Server* sehingga *Server* mendeteksi 0 *Bandwidth*. Pada detik ke-10, *Switch* mulai beradaptasi dengan melakukan *learning* yang menyebabkan *flooding*. Setelah proses *learning* selesai pada detik ke-13, grafik *Bandwidth* kembali normal. Pada detik ke-16 dimana *Link* 1 terdeteksi nyala, secara otomatis *Switch* melakukan *learning* kembali lalu *flooding* yang menyebabkan *Bandwidth* yang diterima oleh *Server* meningkat hingga 400%. Setelah *flooding* selesai yang menandakan *learning* selesai *Bandwidth* kembali seperti normal layaknya percobaan pertama dan kedua.



Gambar 8. Grafik Perbandingan *Bandwidth*

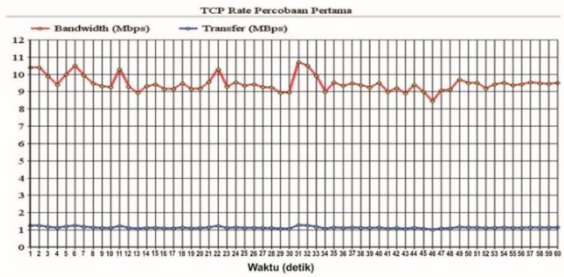
B. *Pengujian Terhadap Paket Transport Control Protocol*

Proses pengujian terhadap paket *TCP* akan dilakukan dengan *tool* ***Iperf*** dari *Host* ke *Server*. *Host* akan mengirim paket *TCP* selama 60 detik melalui *port* tujuan 7777 dengan *Bandwidth* 10Mbps. Informasi tentang percobaan TCP dipaparkan pada Tabel 3 di bawah ini

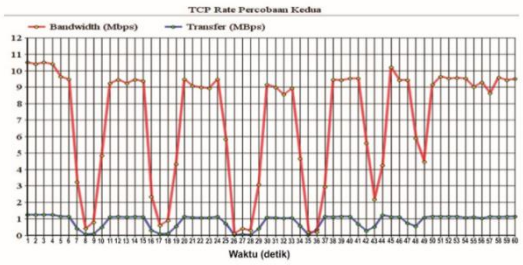
Tabel 3. Tabel Percobaan TCP

No.	Percobaan
1	Metode *Load Sharing* dengan *Weighted Round Robin* sesuai dengan algoritma yang penulis buat.
2	Sama seperti yang pertama namun ditambah apabila terjadi ada *Link* yang mati pada detik ke 6, 15, dan 24. Link aktif pada detik ke 33, 41, dan 47.

Hasil dari pengujian tersebut ditampilkan dengan grafik pada gambar 9, 10, dan 11.



Gambar 9. Grafik TCP *Rate* Percobaan Pertama

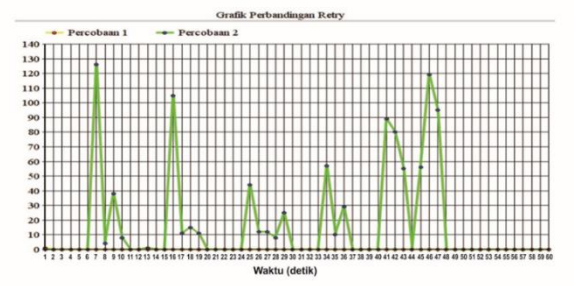


Gambar 10. Grafik TCP *Rate* Percobaan kedua

Dari Gambar 9 terlihat Bandwidth pengiriman TCP dari Host ke Server secara normal menggunakan Load Sharing dengan metode Weighted Round Robin. Gambar 10 ini merupakan grafik dari percobaan kedua.

Dari Gambar 10 terlihat bahwa ada perubahan yang signifikan yang disebabkan event¬-event pada Link yang terhubung ke Server. Pada detik ke-6, Switch mendeteksi Link pertama mati yang menyebabkan ada penurunan Bandwidth. Switch selesai learning untuk event pertama pada detik ke-10 yang menyebabkan Bandwidth kembali seperti semula. Pada detik ke-15, Switch mendeteksi Link kedua mati, dan detik ke-24, Switch mendeteksi Link ketiga mati sehingga terjadi penurunan Bandwidth yang signifikan seperti event pertama. Terutama ketika Link yang tersisa tinggal satu, terlihat Bandwidth pada posisi terendah. Event keempat dimulai pada detik ke-33, dimana Link satu terdeteksi aktif yang menyebabkan Bandwidth juga pada posisi terendah. Selanjutnya di detik ke-41, *Link* ke-2 aktif, dan di detik ke-47 *Link* ketiga kembali aktif. Jadi dari grafik ini terlihat bagaimana *Switch* secara otomatis beradaptasi apabila ada kondisi dimana ada *Link* yang aktif atau mati secara tiba-tiba.

Dari Gambar 11 terlihat bahwa pada saat percobaan pertama hampir tidak ada *retry* sama sekali. *Retry* adalah proses *Server* meminta kembali mengirimkan paket *TCP* dari *Host*. Pada detik ke-6 adalah posisi dimana *retry* tertinggi karena *Link* pertama adalah *Link* yang memiliki bobot tertinggi pada penjadwalan *Weighted Round Robin*. Posisi tertinggi kedua pada detik ke-15 dimana *Link* kedua yang memiliki bobot



tertinggi kedua juga mati. Perubahan yang curam terjadi setiap ada *event* baik ada *Link* yang hidup atau pun mati.

V. KESIMPULAN

Penelitian ini menjelaskan hasil pengujian dari algoritma yang dibuat oleh penulis untuk paket data yang menggunakan protokol UDP. Penulis menggunakan *controller Ryu* dalam mengimplementasikan aplikasi dan emulator M*ininet* untuk mensimulasikan topologi. Hasil dari simulasi menunjukkan bahwa algoritma yang penulis buat mampu menurunkan *jiter* hingga 50% dan paket received out of order berhasil diturunkan hingga 0. Sedangkan waktu yang dibutuhkan untuk kembali *online* ketika ada *link* yang mati adalah 5 detik untuk UDP dan 3 detik untuk TCP. Pada saat pengujian TCP juga terdeteksi 1 Retry dan proses learning untuk fault tolerance dapat dilakukan kurang dari 5 detik.

Dengan Software Defined Networking kita dapat membuat mekanisme baru dari proses yang dilakukan oleh perangkat-perangkat jaringan komputer, dimana hal ini tidak dapat dilakukan pada perangkat-perangkat jaringan komputer konvensional.

DAFTAR PUSTAKA

[1] T.D. Nadeau dan K. Gray, SDN : *Software Defined Networks*, 2013.

[2] US: *Open Networking Foundation*, *Software-defined Networking*: *The New Norm for Networks*. ONF White Paper. Palo Alto, 2013.

[3] H. Long, Y. Shen, M. Guo, dan F. Tang, "*LABERIO: Dynamic Load Balanced Routing in OpenFlow-enabled Networks*". IEEE 27th *International Conference on Advanced Information Networking and Applications*, 2013.

[4] R. Khondoker, A. Zaalouk, R. Marx, dan K. Bayarou, *Feature-based Comparison and Selection of Software Defined Networking (SDN) Controllers*, 2014.

[5] S. Azodomolky, *Software Defined Networking with OpenFlow*.

[6] US: *Open Networking Foundation*. 2012. *Openflow Switch Specification*, 2013.

[7] Sdx Central, *What is Ryu Controller*? https://www.sdxcentral.com/sdn /definitions/sdn-controllers/open-source-sdn-controllers/what-is-ryu controller. Diakses pada tanggal 25 Agustus 2016, 2015.

[8] Ryu SDN Framework Community, *RYU OpenFlow Controller*. https://osrg.github.io/ryu/index.html. Diakses pada tanggal 26 Agustus 2016, 2014.

[9] Mininet Team, *Mininet Overview*. http://mininet.org/overview. Diakses pada tanggal 26 Agustus 2016, 2016.

[10] Bourke, Tony, *Server Load Balancing*. Sebastopol: O'Reilly & Associates, Inc, 2001.
[11] The Linux Foundation, *Bonding*. https://wiki.linuxfoundation.org /networking/bonding. Diakses pada tanggal 20 Desember 2016, 2015.
[12] Cisco, *Link Aggregation Control Protocol* (LACP) (802.3ad) *for Gigabit Interfaces*. http://www.cisco.com/c/en/us/td/docs/ios/12_2sb/feature/guide/gigeth.html. Diakses pada tanggal 25 Desember 2016, 2015.
[13] Wensong, *Job Scheduling Algorithms in Linux Virtual Server*. http://www.linuxvirtualserver.org/docs/scheduling.html. Diakses pada tanggal 19 Desember 2016, 1998.
[14] M. Charles Kozierok, *Transport Layer* (*Layer* 4). http://www.tcpipguide.com/free/t_TransportLayerLayer4.htm. Diakses pada tanggal 26 Desember 2016, 2005.
[15] Ryu Project Team, *Ryu SDN Framework Using OpenFlow 1.3*, 2016.
[16] SDN Central, *Kanazawa University Hospital Reaping the Benefits of a Successful Production SDN Deployment*, 2013.