IMPLEMENTASI METODE *FAILOVER* PADA JARINGAN *SOFTWARE-DEFINED NETWORK*

IMPLEMENTATION OF FAILOVER METHOD ON SOFTWARE-DEFINED NETWORK

Daffa Sidqi Firjatullah, Sofia Naning Hertiana, Ridha Muldina Negara
Prodi S1 Teknik Telekomunikasi, Fakultas Teknik Elektro, Universitas Telkom
daffasidqi06@gmail.com, **sofiananing@telkomuniversity.ac.id**,
ridhanegara@telkomuniversity.ac.id

Abstrak

Jaringan SDN memiliki banyak kelebihan juga memiliki beberapa masalah, salah satu nya yaitu *link failure* atau adanya jalur yang terputus pada saat pengiriman data antar *host*. Komunikasi antara *data plane* dan *control plane* diperlukan untuk mendeteksi, menghitung, dan menyisipkan *rule* yang menciptakan jalur baru. Komunikasi antara *data plane* dengan *control plane* ketika pembangunan rute baru dilakukan menimbulkan terjadinya *latency* atau waktu tunda-antara saat *link failure* terjadi, hal ini juga berdampak pada semua paket yang berpotensi hilang saat terjadi *link failure*. Pembangunan rute baru harus dilakukan agar paket dapat diteruskan ke *data plane*.

Pada tugas akhir ini menerapkan mekanisme *failover* pada jaringan SDN dengan menggunakan fitur *reactive forwarding* dan algoritma *Dijkstra* pada *controller* ONOS dan diterapkan pada topologi, dan parameter uji jaringan yang berbeda. Metode ini menerapkan dua skenario pengujian yang dilakukan yaitu mekanisme *failover* dengan menggunakan *reactive forwarding* dan menggunakan algoritma *Dijkstra*. Masing-masing skenario dilakukan dengan menerapkan pada dua jenis topologi, yaitu *2-D Mesh* dan *Full-Mesh*

Dari percobaan dapat diperoleh bahwa dengan mekanisme *failover*, masalah jalur yang terputus dapat teratasi karena adanya jalur cadangan (*secondary path*). Terdapat perbedaan nilai *Round Trip Time* (RTT), nilai *round trip time* cenderung lebih besar pada skenario dengan menggunakan *reactive forwarding*. Perbedaan nilai *round trip time* dikarenakan pada scenario dengan algoritma *dijkstra*, jalur pengiriman ditentukan oleh ukuran *bandwidth* terbesar dan pada *reactive forwarding* ditentukan oleh *hop count* terkecil. Hasilnya, dengan menggunakan algoritma *Dijkstra* pada saat *bandwidth* yang berbeda tiap *link* dapat menurukan nilai *round trip time* sehingga *latency* menjadi lebih singkat.

Kata Kunci : *Software Defined Network*, ONOS *Controller*, *Failover*, *Latency*, *Round Trip Time*, *Reactive Forwarding*, Algoritma *Dijkstra*

Abstract

SDN has many advantages also has several problems, one of which is a link failure or a disconnected link when sending data between hosts. Communication between the data plane and the control plane is needed to detect, calculate, and insert rules for creating new paths. Communication between data plane and control plane to find a new path when the link failure occurs causes latency or delay, this also potentially affects all packages become lost when the link failure occurs. The construction of the new route is needed to be create so the package can be forwarded to the data plane.

In this experiment applies a failover mechanism on the SDN by using reactive forwarding feature and the Dijkstra's algorithm in ONOS controller, applied to different topologies and network parameter test. This method applies two scenarios, that is failover mechanism using reactive forwarding and using Dijkstra's algorithm. Each scenario is carried out by applying two types of topologies, that is 2-D Mesh and Full-Mesh.

From the experiments it can be obtained that with the failover mechanism, the problem of the disconnected path can be resolved due to a backup path (secondary path). There is a difference in value of Round Trip Time (RTT), round trip time values tend to be greater in scenarios using reactive forwarding. The difference of round trip time values due to in dijkstra's algorithm, the backup path is determined by the biggest bandwidth size and the by

the smallest hop count on reactive forwarding. The result is by using the Dijkstra algorithm when the bandwidth size is different can reduce round trip time values so that the latency become shorter.

Keywords : ***Software Defined Network*, ONOS *Controller, Failover, Latency, Round Trip Time, Reactive Forwarding, Dijkstra's Algorithm*.**

1. Pendahuluan

1.1. Latar Belakang

Software Defined Network atau disebut dengan SDN merupakan paradigma baru dalam dunia jaringan. Konsep dari SDN yaitu memisahkan antara *control plane* dengan *data plane* sehingga pengaturan dapat dilakukan secara terpusat. Perbedaan inilah yang membuat SDN lebih unggul dibandingkan dengan jaringan konvensional pada umumnya. Dalam aristektur SDN terdapat tiga komponen utama yaitu *Application, Control Plane,* dan *Data Plane.* Jaringan SDN yang memiliki banyak kelebihan juga memiliki beberapa masalah, salah satu nya yaitu *link failure* atau adanya jalur yang rusak atau mati pada saat pengiriman data. Komunikasi antara *data plane* dan *control plane* diperlukan untuk mendeteksi, menghitung, dan menyisipkan *rule* yang menciptakan jalur baru. Komunikasi antara *data plane* dengan *control plane* menimbulkan terjadinya *latency* atau waktu tunda-antara saat *link failure* terjadi dan ketika pembangunan rute baru dilakukan. *Latency* dapat berdampak buruk pada aplikasi yang sensitif terhadap waktu dalam jaringan contohnya *live video conference,* juga berdampak pada semua paket yang menjadi hilang saat terjadi *link failure* sebelum pembangunan rute baru dilakukan dan diteruskan ke *data plane* [1].

Pada penelitian R. Ahmed [2] telah dilakukan percobaan mengenai *recovery* atau pemulihan jaringan ketika terjadi *link failure*, yaitu dengan menggunakan mekanisme *failover.* Mekanisme *failover* ini merupakan suatu teknik pada jaringan dengan memberikan dua jalur koneksi (*primary* dan *backup link*) sehingga ketika jalur utama mati, maka koneksi akan tetap berjalan dan akan dialihkan ke jalur cadangan (*backup link*) sehingga masalah *link failure* pada SDN dapat diatasi. Berdasarkan hasil skenario, masih terdapat masalah mengenai *response time* yang dinilai memerlukan waktu yang relatif lama. Pada penelitian M.A Wibowo [3] terdapat metode yang dilakukan untuk mengatasi masalah *link failure* pada jaringan SDN.

Berdasarkan pada permasalahan mengenai *link* yang terputus, pada tugas akhir dilakukan Teknik *failover* dengan menggunakan algoritma *Dijkstra* dan fitur *reactive forwarding* pada *controller* ONOS. Algoritma *Dijkstra* memiliki kelebihan yaitu memiliki informasi pembanding mengenai *link* yang akan dibangun dengan menggunakan bobot atau *cost* yang diterapkan pada tiap *link,* sehingga memiliki *time complexity* yang lebih kecil jika dibandingkan dengan *reactive forwarding* [4].

2. Dasar Teori /Material dan Metodologi/perancangan

2.1 *Software Defined Network* (SDN)

Software Defined Network (SDN) adalah sebuah arsitektur jaringan yang sedang berkembang bersifat dinamis, mudah digunakan, hemat biaya dan *bandwidth,* serta mudah untuk diadaptasi. Konsep dari SDN yaitu memisahkan antara *control plane* dengan *data plane* sehingga pengaturan dapat dilakukan secara terpusat. Perbedaan inilah yang membuat SDN lebih unggul dibandingkan dengan jaringan konvensional pada umumnya. Terdapat tiga komponen utama pada arsitektur SDN yaitu *Application, Control Plane,* dan *Data Plane* [5].

Sebagaimana tercantum pada Gambar 2.1, arsitektur *Software Defined Networking* menurut *Open Network Foundation* (ONF) terdiri dari tiga lapisan yang dapat diakses melalui *Application Programmable Interface* (API)[5]:

1. *Application Layer*, terdiri dari aplikasi end-user yang menggunakan layanan komunikasi *software defined networking*. Batas antara *Application Layer* dan *Control Layer* adalah *northbound* API.
2. *Control Layer* menyediakan fungsionalitas kendali terkonsolidasi yang mengatur proses *forwarding* jaringan melalui antarmuka terbuka.
3. *Infrastructure Layer* terdiri dari elemen jaringan dan perangkat yang menyediakan *switching* dan *forwarding* paket.

2.2. Docker Container

Docker adalah perangkat lunak berbasis *container* yang ringan, mandiri, dan dapat dieksekusi yang mencakup semua dependensi yang diperlukan untuk menjalankan aplikasi, seperti kode, *runtime*, *system tools*, *system library*, dan *settings* [6]

2.3. Mekanisme Failover

Metode *failover* ini merupakan suatu teknik pada jaringan dengan memberikan dua jalur koneksi (*primary* dan *backup link*) sehingga ketika jalur utama mati, maka koneksi akan tetap berjalan dan akan dialihkan ke jalur cadangan (*backup link*) [7]

2.4. Reactive Forwarding

Reactive forwarding adalah sebuah fitur pada *controller* ONOS berfungsi meneruskan atau mengirim atau meneruskan paket berdasarkan *hop* demi *hop* pada setiap paket baru pada jalur cadangan yang bersifat tunggal, fitur ini didasarkan pada nilai *hop count* terkecil dalam pemilihan jalur cadangan [16].

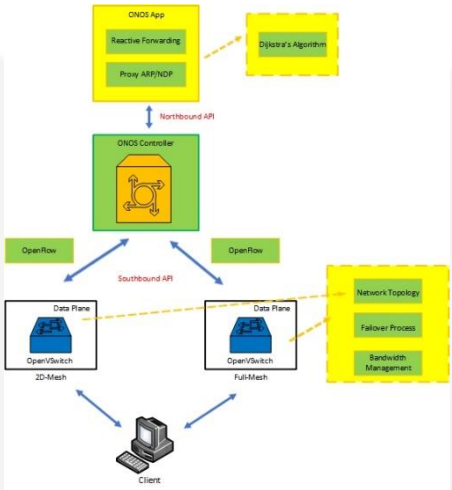
2.4. Algoritma Dijkstra

Algoritma ini bertujuan untuk menemukan jalur terpendek berdasarkan pada bobot terkecil dari satu titik ke titik lain. Dijkstra adalah algoritma yang bersifat *greedy* yang sering digunakan dalam penyelesaian masalah optimasi [8]

3. Peranacangan dan Pengujian Sistem

3.1. Gambaran Umum Sistem

Pada bab ini, akan dibahas mengenai gambaran system secara umum yang dapat dilihat pada gambar 3.1:

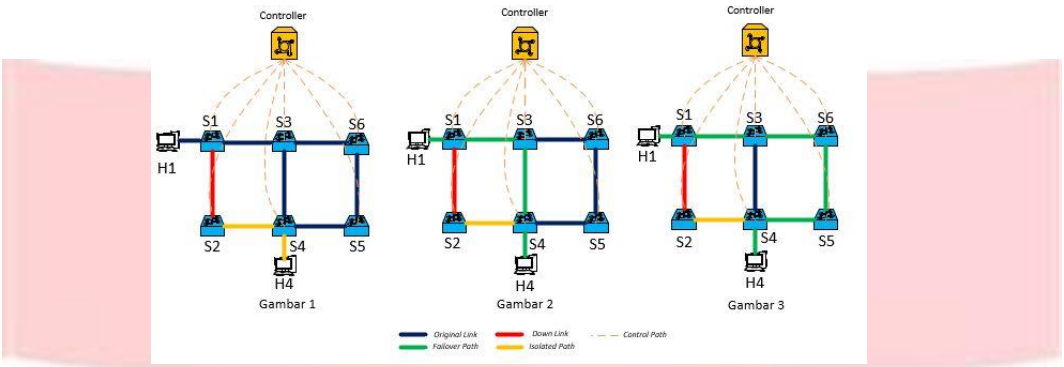


Gambar 3.1 Gambaran Sistem Secara Umum

Gambar 3.1 menjelaskan percobaan yang ingin dilakukan secara garis besar, dan komponen yang dilibatkan adalah sebuah komputer yang membuat jaringan SDN dengan bantuan *emulator* mininet. Saat menjalankan mininet, dibuat *script* topologi jaringan pada *data plane* yang ingin dibentuk, dalam hal ini yaitu topologi *full-mesh* dan 2D-*mesh*, di dalam mininet juga dilakukan instalasi *controller* ONOS sebagai *control plane* atau otak yang mengatur segala aspek dalam jaringan SDN dan dilakukan juga instalasi *openflow* sebagai protocol yang menghubungkan antara *control plane* dengan *data plane*. Pada *data plane* disusun skenario pemutusan *link* pada 2 buah topologi yaitu topologi 2D-*Mesh* dan topologi *Full-Mesh*, dengan menggunakan *controller* ONOS dan menambahkan algoritma *Dijkstra* sebagai algoritma pencarian jalur ketika proses *failover* terjadi.

3.2. Mekanisme *Failover*

Proses *failover* akan melibatkan *control plane* dengan *data plane*. Penjelasan lebih rinci mengenai bagaimana skema *failover* pada SDN akan dijelaskan pada bagian 3.2. Mekanisme *failover* merupakan suatu teknik pada jaringan dengan memberikan dua jalur koneksi (*primary* dan *backup link*) sehingga ketika jalur utama mati, maka koneksi akan tetap berjalan dan akan dialihkan ke jalur cadangan (*backup link*) [7].

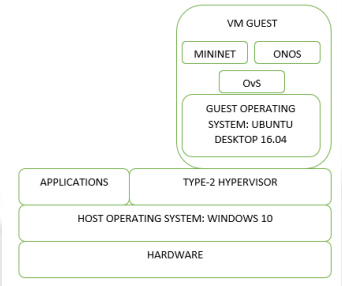


Gambar 3.2 Skema Failover Pada Topologi SDN

Simulasi pada Gambar 3.1 menjelaskan bahwa pada Gambar 3.1 (a), *host* 1 (H1) akan mengirimkan paket berupa ICMP kepada *host* 4 (H4) melalui beberapa *switch* (S) dengan jalur awal yang akan digunakan adalah H1-S1-S2-S4-H4, namun terjadi pemutusan *link* pada Gambar 3.1 (b) (ditandai dengan garis warna merah) antara *switch* 1 (S1) dan *switch 2* (S2) yang menyebabkan *link* antara S2 dan S4 kemudian dari S4 menuju *host* menjadi terisolasi atau tidak dapat dialiri paket (ditandai dengan garis warna kuning). S1 merespon kondisi tersebut dan melakukan komunikasi dengan *controller* diatasnya melalui *control path* (ditandai dengan garis putus-putus warna kuning) untuk menyusun kembali jalur cadangan dan memindahkan jalur tersebut ke jalur cadangan agar paket yang dikirimkan oleh *host* 1 sampai ke tujuan. Jalur cadangan pertama yang disusun oleh *controller* pada Gambar 3.1 (c) yaitu melalui H1-S1-S3-S4-H4 dan jalur cadangan kedua yang disusun dapat dilihat pada Gambar 3.1 (d) yaitu melalui H1-S1-S3-S6-S5-S4-H4.

3.3. Perancangan Kebutuhan Sistem

Pada bagian ini, akan dibahas mengenai perancangan sistem yang akan diuji. Sistem akan dibangun diatas sebuah *Virtual Machine* (VM) Ubuntu 16.04 yang diinstal pada *hypervisor* Oracle VM VirtualBox di atas *host.* Arsitektur sistem dapat dilihat pada Gambar 3.1 dibawah ini.

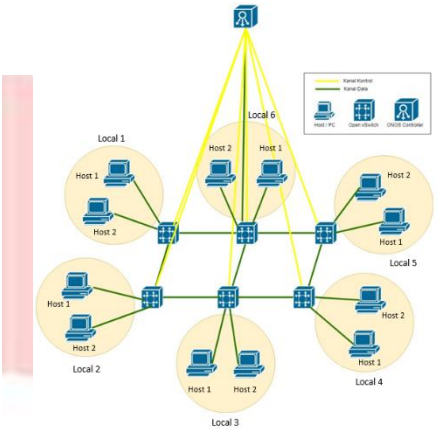


Gambar 3.3 Perancangan Kebutuhan Sistem

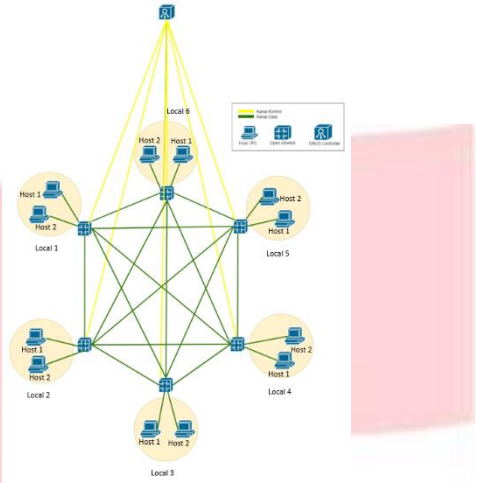
Seperti yang terlihat pada gambar, komputer *host* menggunakan sistem operasi Windows 10 *Home*. Pada komputer *host* diinstal perangkat lunak *hypervisor* tipe-2 yaitu Oracle VirtualBox. Di atas *hypervisor* tersebut akan dibuat VM *Guest* dengan sistem operasi Ubuntu Desktop 16.04 dengan spesifikasi seperti yang tertera pada Tabel 3.1 dibawah ini. Pada VM *Guest* ini dilakukan instalasi *controller* ONOS, *emulator* Mininet, dan Open v*Switch*

3.4. Desain Topologi Jaringan

Tugas akhir ini akan menggunakan simulasi dua jaringan dengan dua topologi jaringan yang berbeda sebagai perbandingan, yaitu topologi *2-D Mesh* dan *Full Mesh.* Pada penelitian tugas akhir ini, terdapat dua skenario pengujian yang akan dilakukan. Masing-masing skenario akan dilakukan sebanyak dua puluh kali pada masing-masing topologi dengan pemutusan *link* antar *switch* yang bervariasi agar mendapatkan hasil yang akurat. Hasil pengujian dari kedua topologi selanjutnya akan dianalisis dan dibandingkan.



Gambar 3.4 Topologi 2-D Mesh



Gambar 3.5 Topologi Full-Mesh

3.5. Skenario Pengujian

Terdapat dua skenario pengujian yang dilakukan, masing-masing skenario dilakukan variasi pemutusan *link* antar *switch* pada dua jenis topologi, yaitu *2-D Mesh* pada Gambar 3.4 dan *Full-Mesh* pada Gambar 3.5.

Hal pertama yang dilakukan adalah menyusun *script python* pada topologi yang digunakan dan mengatur besaran *bandwidth* pada masing-masing *link*. Daftar ini berguna pada skenario pengujian dengan ukuran *bandwidth* yang berbeda. Daftar besaran *bandwidth* pada tiap-tiap *link* pada topologi 2D-*mesh* dan *full-mesh* dapat dilihat pada Tabel 3.3 dan 3.4 dibawah ini:

Tabel 3.1 Tabel Besaran *Bandwidth* Tiap *Link* Topologi 2D-*Mesh*

No	*Links*	*Bandwidth*
1.	S1-S2	1000 Mbit/sec
2.	S1-S6	1000 Mbit/sec
3.	S2-S3	1000 Mbit/sec
4.	S2-S5	1000 Mbit/sec

No	*Links*	*Bandwidth*
5.	S3-S4	1000 Mbit/sec
6.	S4-S5	100 Mbit/sec
7.	S5-S6	100 Mbit/sec

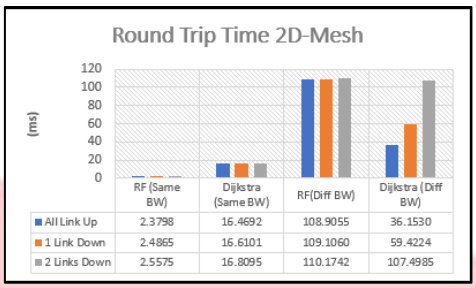
Tabel 3.2 Tabel Besaran *Bandwidth* Tiap *Link* Topologi *Full-Mesh*

No	*Links*	*Bandwidth*
1.	S1-S2	100 Mbit/sec
2.	S1-S3	1000 Mbit/sec
3.	S1-S4	100 Mbit/sec
4.	S1-S5	1000 Mbit/sec
5.	S1-S6	10 Mbit/sec
6.	S2-S3	100 Mbit/sec
7.	S2-S4	100 Mbit/sec
8.	S2-S5	1000 Mbit/sec

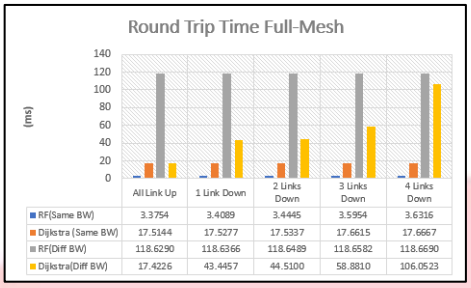
No.	*Links*	*Bandwidth*
9.	S2-S6	1000 Mbit/sec
10.	S3-S4	1000 Mbit/sec
11.	S3-S5	10 Mbit/sec
12.	S3-S6	10 Mbit/sec
13.	S4-S5	1000 Mbit/sec
14.	S4-S6	100 Mbit/sec
15.	S5-S6	1000 Mbit/sec

3.5.1. Pengujian *Round Trip Time* Pada *Failover*

Pada pengujian ini dilakukan uji *failover* menggunakan fitur *reactive forwarding* pada ONOS *Application* dan diterapkan dengan dua skenario pada kedua topologi, yaitu menggunakan *bandwidth* dengan ukuran berbeda sesuai dengan tabel 3.3 dan 3.4, dan dibuat sama yaitu sebesar 1000 Mbps[17]. Percobaan dilakukan dengan mengirimkan paket icmp dari *host* sumber ke *host* tujuan selama 60 detik, kemudian dilakukan sebanyak 30 kali, kemudian dilakukan pemutusan *link* yang jumlahnya bervariasi, dan didapatkan data *Round Trip Time* (RTT) dari pengiriman paket icmp tersebut.



Gambar 3.10 Nilai Pada Topologi 2D-*Mesh*



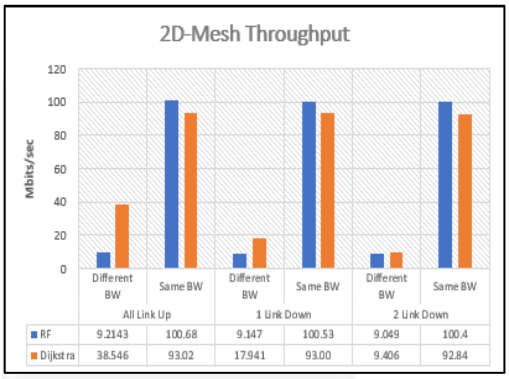
Gambar 3.11 Nilai Pada Topologi *Full-Mesh*

Nilai *round trip time* pada *reactive forwarding* akan cenderung lebih besar jika dibandingkan dengan nilai RTT pada algoritma *Dijkstra*. Hal ini diakibatkan pada algoritma *Dijkstra* dilakukan pemilihan jalur berdasarkan ukuran *bandwidth* yang terbesar.

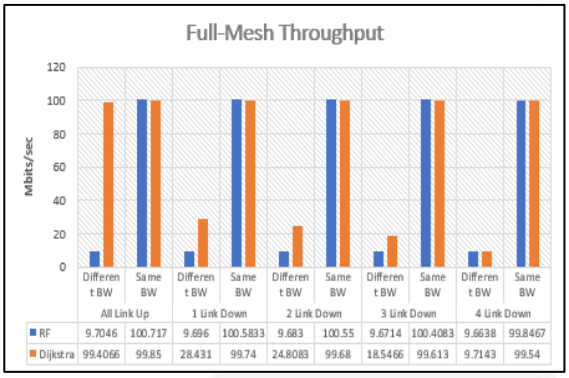
3.5.2. Pengujian Parameter Uji Jaringan

Parameter yang diuji yaitu *throughput, jitter,* dan *packet loss*. Pengujian dilakukan pada kedua jenis topologi yaitu topologi 2D-*mesh* dan topologi *full-mesh*. Percobaan dilakukan dengan membangkitkan trafik UDP menggunakan *tools* iperf secara bergantian pada tiap *host* selama satu menit, skenario pemutusan *link* yang digunakan serupa seperti skenario pada pengujian *round trip time* . Kemudian hasil yang diperoleh akan dihitung nilai rata-rata nya.

3.5.2.1. Pengujian *Throughput*



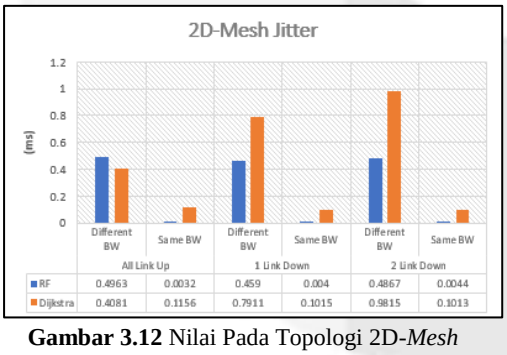
Gambar 3.10 Nilai Pada Topologi 2D-*Mesh*



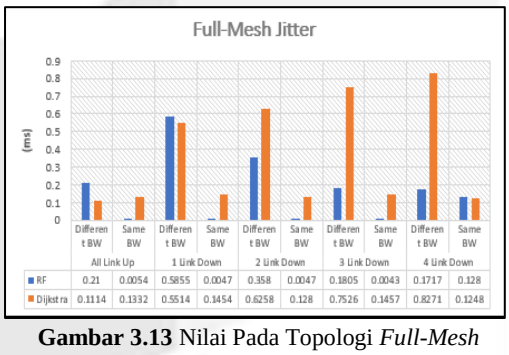
Gambar 3.11 Nilai Pada Topologi *Full-Mesh*

Nilai *throughput* pada kedua topologi diatas mengalami penurunan, hal ini beriringan dengan jumlah paket yang diterima di sisi *client* dan berhubungan dengan besaran nilai dari parameter performansi lainnya yaitu *packet loss*, semakin tinggi *packet loss*, maka semakin rendah *throughput* yang didapat.

3.5.2.2. Pengujian *Jitter*



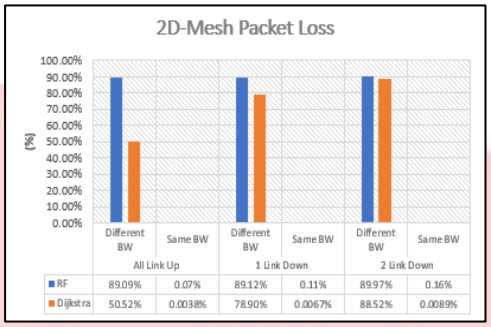
Gambar 3.12 Nilai Pada Topologi 2D-*Mesh*



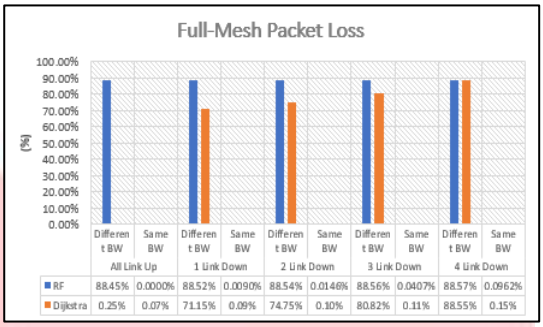
Gambar 3.13 Nilai Pada Topologi *Full-Mesh*

Perbedaan nilai *jitter* menunjukkan besaran variansi dari waktu *delay* saat pengiriman paket UDP berlangsung, selain itu juga diakibatkan oleh jalur yang dilewati oleh paket berbeda-beda.

3.5.2.3. Pengujian *Packet Loss*



Gambar 3.14 Nilai Pada Topologi 2D-*Mesh*



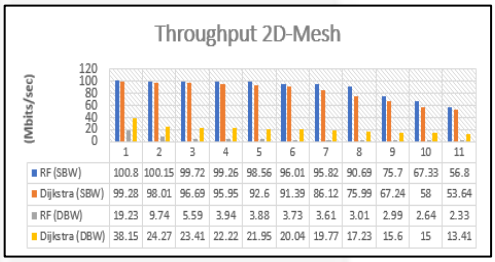
Gambar 3.15 Nilai Pada Topologi *Full-Mesh*

Nilai *packet loss* disebabkan oleh adanya paket yang dibuang pada saat pengiriman. Dapat disebabkan oleh beberapa factor, salah satunya yaitu paket yang *corrupt* dan disebabkan juga oleh *variasi bandwidth* yang diterapkan pada masing-masing *link* yang berbeda-beda, sementara trafik yang dibangkitkan selalu sama yaitu 1000 Mbit/s. Penyempitan ukuran *bandwidth* mengakibatkan terdapat paket yang dibuang pada saat pengiriman data.

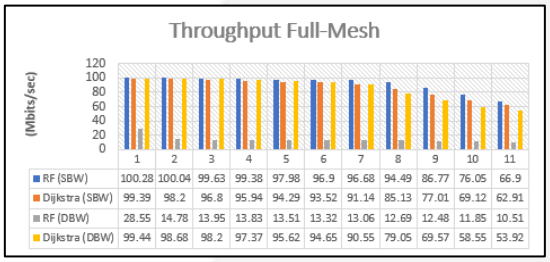
3.5.3. Pengujian Pemenuhan Trafik

Pengujian ini dilakukan dengan mengirimkan trafik dari masing-masing *host* yang berangsur-angsur bertambah berukuran 100 Mbyte pada kedua topologi yang berbeda menggunakan *tools* iperf dan dilakukan selama 60 detik dan dicari nilai rata-ratanya.

3.5.3.1. Pengujian *Throughput*



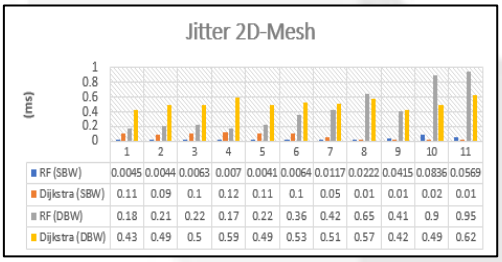
Gambar 3.16 Nilai Pada Topologi 2D-*Mesh*



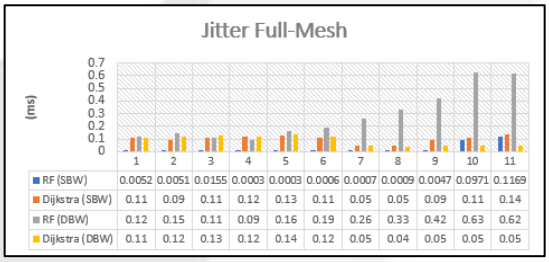
Gambar 3.17 Nilai Pada Topologi *Full-Mesh*

Nilai *throughput* pada kedua topologi berangsur-angsur turun seiring kenaikan jumlah *host* yang mengirimkan trafik.

3.5.3.2. Pengujian *Jitter*



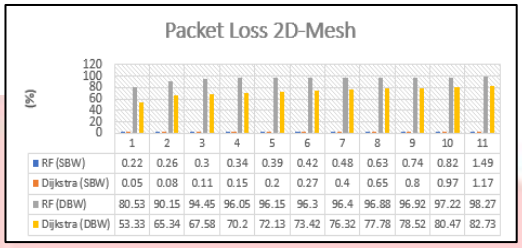
Gambar 3.18 Nilai Pada Topologi 2D-*Mesh*



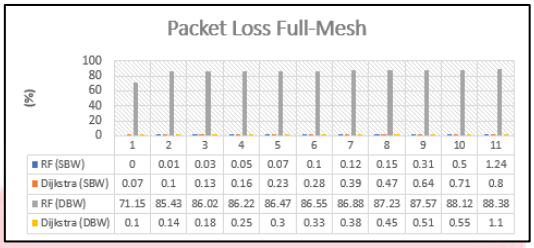
Gambar 3.19 Nilai Pada Topologi *Full-Mesh*

Nilai *jitter* pada kedua topologi bervariasi, semakin kecil nilai *jitter* maka jaringan tersebut dikatakan stabil.

3.5.3.3. Pengujian *Packet Loss*



Gambar 3.20 Nilai Pada Topologi 2D-*Mesh*



Gambar 3.21 Nilai Pada Topologi *Full-Mesh*

Nilai *packet loss* pada kedua topologi diatas berangsur-angsur naik seiring dengan bertambahnya jumlah *host* yang mengirimkan trafik.

4. Kesimpulan

Dari hasil pengamatan yang menunjukkan bahwa nilai *round trip time* yang menurun saat menggunakan algoritma *dijkstra*, dapat disimpulkan bahwa penggunaan algoritma *Dijkstra* dalam mekanisme *failover* terutama pada saat ukuran *bandwidth* di setiap *link* yang berbeda-beda pada topologi dapat mengurangi *latency* dibandingkan dengan *reactive forwarding*, sehingga pengiriman data menjadi lebih singkat.

Daftar Pustaka:

[1] Ryan Izard, "How to Work with Fast-Failover OpenFlow Groups - Floodlight Controller - Project Floodlight," 2018. [Online]. Available: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/7995427/How+to+Work+with+Fast-Failover+OpenFlow+Groups#HowtoWorkwithFast-FailoverOpenFlowGroups-OpenFlowGroups. [Accessed: 28-Feb-2019].

[2] R. Ahmed, E. Alfaki, and M. Nawari, "Fast failure detection and recovery mechanism for dynamic networks using software-defined networking," *Proc. 2016 Conf. Basic Sci. Eng. Stud. SGCAC 2016*, pp. 167–170, 2016.

[3] M. A. Wibowo, W. Yahya, and D. P. Kartikasari, "Implementasi Link Fast-Failover Pada Multipath Routing Jaringan Software-Defined Network," *J. Pengemb. Teknol. Inf. dan Ilmu Komput. Univ. Brawijaya*, vol. 2, no. 10, pp. 3968–3975, 2018.

[4] Y. Retanto, "Algoritma Dijkstra dan Bellman-Ford dalam Pencarian Jalur Terpendek," *Makal. If2091*, 2009.

[5] O. N. F. Solution, B. September, and ONF, "SDN in the Campus Environment," *ONF Work.*, 2013.

[6] "What is OpenFlow? Definition and How it Relates to SDN." [Online]. Available: https://www.sdxcentral.com/networking/sdn/definitions/what-is-openflow/.

[7] "OpenFlow architecture | Download Scientific Diagram." [Online]. Available: https://www.researchgate.net/figure/OpenFlow-architecture_fig1_272579006.

[8] onosproject.org, "Software Defined Networking (SDN) Introducing ONOS-a SDN network operating system for Service Providers."

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

[10] "Mininet: An Instant Virtual Network on your Laptop (or other PC) - Mininet." [Online]. Available: http://mininet.org/.

[11] "Production Quality, Multilayer Open Virtual Switch." [Online]. Available: https://www.openvswitch.org/.

[12] I. Sofana, *Teori dan Praktik Cloud Computing*, 1st ed. Bandung: Informatika Bandung, 2012.

[13] "What is a Container? | App Containerization | Docker." [Online]. Available: https://www.docker.com/resources/what-container. [Accessed: 14-Oct-2019].

[14] "PuTTY - a free SSH and telnet client for Windows." [Online]. Available: https://www.putty.org/. [Accessed: 13-May-2019].

[15] R. M. N. H.A. Naqvi, S.N. Hertiana, "Simulasi dan Analisis Sistem Gateway Terdistribusi untuk First-Hop Redundancy dan Load Balancing," *e-Poceeding Eng.*, vol. 2, no. 2, pp. 2729–2736, 2015.

[16] B. K. Thapa and B. Dikici, "Reactive Forwarding Applications in ONOS," no. January, 2018.

[17] Andi Kristanto, *Jaringan Komputer*. Yogyakarta: Graha Ilmu, 2003.

[18] T. Irfan, "opickers90/ONOS-ISP.v.1." [Online]. Available: https://github.com/opickers90/ONOS-ISP.v.1. [Accessed: 24-Oct-2019].