



INSTITUT TEKNOLOGI DEL

***Comparative Study of Singlepath and Multipath Routing in Terms of
Network Performance and Designing SDN Testbed***

TUGAS AKHIR

Diajukan sebagai salah satu syarat untuk memperoleh gelar Diploma III
Program Studi Diploma III Teknologi Komputer

13317002

Wendi Martin Situmeang

13317006

Ade Kurniawan

**FAKULTAS INFORMATIKA DAN TEKNIK ELEKTRO
PROGRAM STUDI DIPLOMA 3 TEKNOLOGI KOMPUTER
INSTITUT TEKNOLOGI DEL**

Agustus 2020



INSTITUT TEKNOLOGI DEL

***Comparative Study of Singlepath and Multipath Routing in Terms of
Network Performance and Designing SDN Testbed***

TUGAS AKHIR

13317002

Wendi Martin Situmeang

13317006

Ade Kurniawan

**FAKULTAS INFORMATIKA DAN TEKNIK ELEKTRO
PROGRAM STUDI DIPLOMA 3 TEKNOLOGI KOMPUTER
INSTITUT TEKNOLOGI DEL**

Agustus 2020

HALAMAN PERNYATAAN ORISINALITAS

Tugas Akhir ini adalah hasil karya saya sendiri, dan semua sumber baik yang dikutip maupun dirujuk telah saya nyatakan dengan benar.

NAMA : Wendi Martin Situneang

NIM : 13317002

TANDA TANGAN :

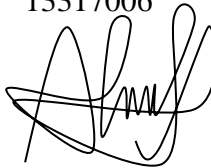


TANGGAL : Agustus 2020

NAMA : Ade Kurniawan

NIM : 13317006

TANDA TANGAN :



TANGGAL : Agustus 2020

HALAMAN PENGESAHAN

Tugas Akhir ini diajukan oleh :

1. Nama : Wendi Martin Situmeang

NIM : 13317002

Program studi : D3-Teknologi Komputer

2. Nama : Ade kurniawan

NIM : 13317006

Program studi : D3-Teknologi Komputer

Judul Tugas Akhir : *Comparative Study of Singlepath and Multipath Routing in Terms of Network Performance and Designing SDN Testbed*

Telah berhasil dipertahankan dihadapannya dewan penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Diploma III pada program studi Diploma III Teknologi Komputer Fakultas Informatika dan Teknik Elektro Institut Teknologi Del.

DEWAN PENGUJI

Pembimbing I : Eka Stephani Sinambela, S.ST.,M.Sc. ()

Pembimbing II: Deni P.Lumbantoruan, S.T.,M.Eng ()

Penguji I: Sari Muthia Silalahi, S.Pd.,M.Ed ()

Penguji II: Istas Manalu, S.Si.,M.Sc ()

Ditetapkan : Laguboti
Tanggal : 04 Agustus 2020

KATA PENGANTAR

Puji dan syukur kepada Tuhan Yang Maha Esa, atas rahmat yang diberikan kepada penulis sehingga Tugas Akhir ini dapat diselesaikan dengan baik. Laporan Tugas Akhir ini bertujuan untuk memberikan informasi bagi pembaca mengenai *Implementing Comparative study of singlepath and multipath routing in terms of network performance and designing SDN testbed*. Laporan Tugas Akhir ini merupakan syarat kelulusan Diploma III Institut Teknologi Del.

Penulis menyampaikan terima kasih kepada berbagai pihak, khususnya kepada:

1. Orangtua, saudara, dan teman dekat yang telah memberikan dukungan dalam pengerjaan Tugas Akhir.
2. Eka Stephani Sinambela,S.ST.,M.Sc dan Deni Lumbantoruan,S.T,M.Eng selaku dosen pembimbing tugas akhir yang telah memberikan saran dan arahan dalam menyelesaikan tugas akhir dan banyak meluangkan waktu dan tenaga untuk membimbing penulis selama penyusunan tugas akhir.
3. Istas Manalu,S.Si.,M.Sc dan Sari Muthia Silalahi,S.Pd.,M.Ed sebagai dosen penguji Tugas Akhir
4. Bapak/Ibu dosen khususnya yang mengajar di Jurusan Teknologi Komputer yang telah membekali penulis dengan beberapa disiplin ilmu yang berguna.
5. Teman-teman seperjuangan Mahasiswa Teknologi Komputer Institut Teknologi Del Angkatan 2017 yang telah banyak berdiskusi dan bekerjasama dengan penulis selama masa pendidikan.

Penulis menyadari bahwa pada tugas akhir ini masih memiliki kekurangan dan kelemahan. Oleh karena itu, penulis mengharapkan kritik dan saran yang membangun agar tugas akhir yang dibuat menjadi lebih baik kedepannya. Semoga keberadaan tugas akhir ini dapat bermanfaat dan menambah wawasan bagi semua pihak, khususnya tentang *Comparative study of singlepath and multipath routing in terms of network performance and designing SDN testbed*. Akhir kata penulis mengucapkan terimakasih.

Laguboti, Agustus 2020

Wendi Martin Situmeang

Ade Kurniawan

**HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI
TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS**

Sebagai sivitas akademik Institut Teknologi Del, saya yang bertanda tangan di bawah ini:

Nama : Wendi Martin Situmeang
NIM : 13317002
Fakultas/Program Studi : FITE/D3 Teknologi Komputer

Nama : Ade Kurniawan
NIM : 13317006
Fakultas/Program Studi : FITE/D3 Teknologi Komputer

Jenis Karya : Tugas Akhir

Demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Institut Teknologi Del Hak Bebas Royalti Noneksklusif (*Non-exclusive Royalty-Free Right*) atas karya ilmiah saya yang berjudul:

***Comparative Study of Singlepath and Multipath Routing in Terms of Network
Performance and Designing SDN Testbed***

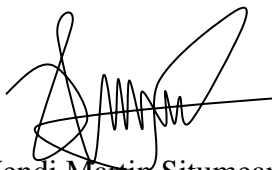
Beserta perangkat yang ada (jika di perlukan). Dengan Hak Bebas Royalti Noneksklusif ini Institut Teknologi Del berhak menyimpan, mengalih/media-format dalam bentuk pangkalan data (database), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik hak cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

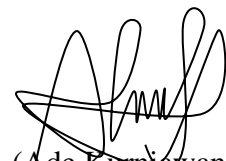
Ditetapkan di : Laguboti

Pada tanggal : 04 Agustus 2020

Yang menyatakan



(Wendi Martin Situmeang)



(Ade Kurniawan)

ABSTRAK

Program Studi : DIII Teknologi Komputer
Judul : *Comparative study of singlepath and multipath routing in terms of network performance and designing SDN testbed*

Routing adalah suatu proses untuk mengirimkan suatu paket data dalam sebuah jaringan komputer dari node asal sampai dengan node tujuan. Tujuan dari *routing* adalah untuk menemukan rute terbaik untuk mengirimkan atau mentransmisikan paket data. Untuk mendapatkan rute, maka algoritma dan teknik *routing* terus dikembangkan untuk mendapatkan hasil paling optimal. Sebagian besar protokol *routing* yang digunakan saat ini menggunakan algoritma yang menghasilkan jalur tunggal saja, sehingga ketika melakukan pengiriman paket data pada siklus trafik yang cenderung tinggi dapat menyebabkan terjadinya kemacetan (*bottleneck*) dan tabrakan antar paket (*congestion*) yang berdampak kepada pemborosan sumber daya jaringan. Dalam pengerjaan Tugas Akhir ini, Tugas Akhir ini menggunakan Konsep *Software Defined Network* (SDN) dan teknik *multipath routing* dalam implementasi sistem jaringan. SDN merupakan arsitektur jaringan yang memiliki konsep dimana sistem control plane dan *forwarding/data plane* dipisahkan sehingga jaringan yang dikelola lebih dinamis dan *programable*. *Multipath routing* dengan menggunakan algoritma ECMP digunakan untuk melakukan pengiriman paket data sehingga dapat meningkatkan *throughput* dan dapat melakukan *load balancing* karena menggunakan beberapa jalur dalam pengirimannya. Setelah dilakukan implementasi *multipath routing* di SDN menggunakan algoritma ECMP maka dilakukan pengujian dengan membandingkan hasil pengiriman paket data antara *multipath routing* dan *singlepath routing*. Iperf digunakan sebagai tools untuk mengukur nilai *throughput*, *delay* dan *paket loss* ketika paket data dikirimkan dari *source* ke *destination*. Kemudian hasil dari nilai *throughput*, *delay* dan *paket loss* dijadikan sebagai pembanding kinerja jaringan antara *multipath routing* dan *singlepath routing*. Dari pengujian yang telah dilakukan dengan mengukur *throughput*, *delay* dan *paket loss* pada *multipath routing* di SDN kemudian dilakukan perbandingan dengan *singlepath routing*, maka didapatkan kesimpulan bahwa implementasi *multipath routing* in SDN testbed dapat meningkatkan performa jaringan.

Kata Kunci : *Routing, Software defined network(SDN), Multipath routing, ECMP, Iperf*

ABSTRACT

Program Studi : DIII Teknologi Komputer
Judul : *Comparative study of singlepath and multipath routing in terms of network performance and designing SDN testbed*

Routing is a process to move a data packet in a computer network from the origin node to the destination node. The purpose of routing is to find the best route for sending or transmitting data packets. Algorithms and routing techniques continue to be developed to get the most optimal results and reduce network burden when transmitting data. Most of the routing protocols used today use algorithms that produce a *singlepath* so that when sending data packets on a traffic cycle that tends to be high it can cause bottlenecks and collisions between packets (congestion) that have an impact on the waste of network resources. In this final project, the author uses the concept of Software Defined Network (SDN) and *multipath* routing techniques in the implementation of network systems. SDN is a network architecture that has a concept where the control plane and forwarding / data plane systems are separated so that the managed network is more dynamic and programmable. *Multipath* routing using the ECMP algorithm is used to send data packets so that it can increase throughput and load balancing because it uses multiple paths in sending. After implementing *multipath* routing on SDN using the ECMP algorithm, testing is done by comparing the results of sending data packets between *multipath* routing and *singlepath* routing. Iperf is used as a tool to measure the value of throughput, delay and packet loss when data packets are sent from source to destination. Then the results of the value of throughput, delay and packet loss are used as a comparison of network performance between *multipath* routing and *singlepath* routing. From the testing that has been done by measuring throughput, delay and packet loss in *multipath* routing at SDN then comparing with *singlepath* routing, it is concluded that implementing *multipath* routing in SDN testbed can improve network performance.

Keywords : *Routing, Software defined network(SDN), Multipath routing, ECMP, Iperf*

DAFTAR ISI

KATA PENGANTAR	iv
ABSTRAK	vi
BAB I PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Research Question	2
1.3 Tujuan	2
1.4 Lingkup	2
1.5 Pendekatan	2
1.6 Sistematika Penyajian	3
1.7 Istilah dan Definisi	4
BAB II TINJAUAN PUSTAKA	5
2.1 Landasan Teori	5
2.1.1 Software Define Network	5
2.1.2 <i>Singlepath Routing</i>	6
2.1.3 <i>Multipath Routing</i>	6
2.1.4 <i>Openflow</i>	7
2.1.5 Ryu	9
2.1.6 ICMP (Internet Control Message Protocol)	9
2.1.7 Iperf	10
2.1.8 Mininet	11
2.1.9 Metrik	11
2.2 Related Work	13
2.2.1 SDN Implementation of <i>Multipath</i> Discovery to Improve Network Performance in Distributed Storage Systems	13
2.2.2 HiQoS: An SDN-Based <i>Multipath</i> QoS Solution	13
2.2.3 <i>Multipath Routing</i> in SDN for Network Performance Improvement	14
BAB III ANALISIS DAN PERANCANGAN	15
3.1. Analisis	15
3.1.1 Analisis Masalah	15
3.1.2 Analisis Pemecahan Masalah	15
3.1.3. Analisis Kebutuhan Sistem	16
3.1.3.1 Kebutuhan Perangkat Keras	16
3.2 Perancangan Pembangunan Sistem	17
3.2.1 Sistem Arsitektur	18
3.2.2 <i>Multipath routing</i> dan QoS Management	20
3.2.3 Perancangan Perangkat Keras	20
3.2.4 Flowchart Sistem	22
3.3 Skenario Pengujian	25
3.3.1 Pengujian <i>Multipath Routing</i> dalam No-limited Bandwidth Topologi	25
3.3.2 Pengujian QoS dalam Limited Bandwidth Topologi	26
BAB IV IMPLEMENTASI DAN PENGUJIAN	27
4.1 Implementasi dan Pengujian <i>Singlepath</i> dengan <i>Multipath routing</i> di Simulator	27
4.1.1 instalasi mininet	27
4.1.2 Instalasi Ryu di simulator	27
4.2 Pengujian SDN di Mininet	28
4.2.1 <i>Singlepath Routing</i> SDN di Mininet	28
4.2.2 <i>Multipath Routing</i> SDN di Mininet	32

4.3 Pengujian Antara <i>Singlepath</i> dengan <i>Multipath routing</i>	36
4.3.1 Hasil Perbandingan Pengujian Antara <i>Singlepath</i> dengan <i>Multipath routing</i>	39
4.4 Perancangan Testbed pada Raspberry Pi	41
4.4.1 Instalasi Raspberry Pi	41
4.4.2 Instalasi <i>OpenVswitch</i> di Raspberry Pi	42
4.4.3 Instalasi Ryu di Raspberry Pi	43
Bab 5 KESIMPULAN DAN SARAN	45
5.1 Kesimpulan.....	45
5.2 Saran	45
Daftar Pustaka	i
LAMPIRAN.....	iii

DAFTAR TABEL

Tabel 1. Istilah.....	4
Tabel 2. Daftar Singkatan	4
Tabel 3. Kebutuhan Perangkat Keras.....	16
Tabel 4. Kebutuhan Perangkat Lunak	17
Tabel 5. Flowtable	19
Tabel 6. Keterangan Topologi Miniedit.....	29
Tabel 7. Skenario 1	36
Tabel 8. Skenario 1.1.....	37
Tabel 9. Skenario 2	37
Tabel 10. Skenario 2.2	38

DAFTAR GAMBAR

Gambar 1. Arsitektur SDN	5
Gambar 2. Sistem Kerja <i>Openflow</i>	7
Gambar 3. Raspberry Pi	8
Gambar 4. Diagram Iperf	10
Gambar 5. Arsitektur <i>Control Plane</i> dan <i>Data Plane</i>	18
Gambar 6. Openflow	19
Gambar 7. Rancangan Umum	21
Gambar 8. Flowchart secara umum	22
Gambar 9. Flowchart <i>OpenVswitch</i>	23
Gambar 10. Flowchar <i>Controller</i>	23
Gambar 11. Flowchart Algoritma ECMP	24
Gambar 12. Graf ECMP	24
Gambar 13. Topologi <i>Singlepath routing</i>	28
Gambar 14. Topologi <i>Multipath routing</i>	32
Gambar 15. Grafik perbandingan <i>singlepath</i> dengan <i>Multipath</i>	39
Gambar 16. Host Delay di <i>Multipath</i> Routing dan <i>Singlepath</i>	40
Gambar 17. Paket Loss	41

BAB I PENDAHULUAN

Pada bab ini berisi penjelasan mengenai latar belakang, tujuan pelaksanaan, ruang lingkup, pendekatan, istilah, definisi dan sistematika penyajian Tugas Akhir.

1.1 Latar Belakang

Dalam pengiriman paket dari *host* sumber menuju *host* destinasi melalui komunikasi antar-jaringan pada umumnya dilakukan dengan menggunakan *routing* protokol yang berfungsi untuk merutekan jalur terbaik yang harus dilalui oleh paket untuk sampai ke tujuan. *Routing* protokol yang secara umum digunakan adalah jenis jalur tunggal (*singlepath*). Pada implementasinya, *singlepath routing* dapat menentukan jalur optimal yang akan ditempuh antar *host* dengan memperhitungkan metrik jarak, delay dan *bandwidth* [1]. Meskipun demikian, ditemukan kelemahan pada *singlepath routing* yaitu penggunaan jalur tunggal yang dapat menyebabkan terjadinya kemacetan (*bottleneck*) dan tabrakan antar paket (*congestion*) jika trafik tinggi. Kondisi ini berdampak pada *throughput* jaringan yang menurun bahkan paket yang dikirimkan dapat di-*drop*. Kelemahan lainnya, *singlepath* tidak dapat memfasilitasi trafik data dari aplikasi tertentu yang membutuhkan prioritas *Quality of Service* (QoS) yang handal [2].

Multipath routing merupakan pendekatan alternatif yang dapat digunakan untuk menyelesaikan permasalahan yang ditimbulkan dari penggunaan *singlepath routing*. Peningkatan performa jaringan dengan *Multipath routing* dicapai melalui penggunaan beberapa jalur secara bersama selama proses transmisi data berlangsung. *Multipath routing* juga menggunakan metode *congestion control* untuk menentukan seberapa banyak trafik yang dilewatkan pada setiap jalur. Pendekatan ini membuat penggunaan sumber daya jauh lebih efektif dan meningkatkan *throughput* jaringan [1].

Untuk semakin meningkatkan performansi *Multipath routing* didukung dengan menggunakan sebuah arsitektur jaringan yang baru yaitu *Software Defined Networking* (SDN). SDN muncul dilatarbelakangi oleh perkembangan internet dan teknologi jaringan yang menyebabkan produksi dari perangkat jaringan seperti *switch* dan *router* semakin bervariasi. Pada sebuah jaringan tradisional dalam skala besar terdapat berbagai macam perangkat jaringan yang diproduksi oleh vendor yang berbeda. Setiap vendor akan menerapkan *rule* dan fungsi *routing* yang berbeda pada setiap perangkat sehingga jaringan terlihat sangat kompleks, tidak efisien untuk dikelola, sulit untuk mendukung skalabilitas dan memerlukan biaya yang besar untuk mengadakan jenis perangkat yang berbeda [3].

Keunggulan utama menggunakan SDN adalah dengan adanya kontroler atau *control plane* maka *flow table* yang digunakan pada konsep *routing* dan juga topologi seluruh jaringan dapat direkam dengan baik. Kontroler akan fokus untuk mengatur aliran (*flow*) paket secara logik dan menempatkan *flow table* tersebut pada perangkat *routing* lainnya. Sedangkan, *data plane* akan fokus untuk mengalirkan paket menuju perangkat jaringan lainnya sesuai dengan *flow* yang diberikan oleh kontroler [4]. Dengan keunggulan SDN ini, *Multipath* pada *Multipath routing* dapat dihitung secara efisien untuk mengoptimalkan penempatan *flow table* pada perangkat *routing* lainnya.

Penelitian ini dijalankan untuk memberi kontribusi dalam mengimplementasikan simulasi yang telah dilakukan sebelumnya dengan membentuk rancangan testbed SDN. Rancangan testbed SDN akan dibentuk dengan menggunakan *Raspberry Pi* yang akan berperan sebagai *Open vSwitch* dan sebuah *Controller*. Selanjutnya, pada *testbed* ini performansi *Multipath*

routing akan diuji kembali dengan ekspektasi akan menghasilkan kinerja yang signifikan antara simulasi dengan implementasi pada *testbed*.

1.2 Research Question

1. Bagaimana performa jaringan dapat ditingkatkan dengan menggunakan *multipath routing* pada *platform SDN*?
2. Bagaimana rancangan testbed SDN dengan menggunakan perangkat *Raspberry Pi* sebagai *OpenVswitch* dan kontroler untuk mengimplementasikan *Multipath routing* ?

1.3 Tujuan

Adapun tujuan dari Tugas Akhir ini adalah:

1. Untuk mengetahui peningkatan performa jaringan yang mengimplementasikan *Multipath routing* di platform *SDN*. Kinerja *multipath routing* akan dibandingkan dengan *singlepath routing* dengan menggunakan simulator.
2. Menghasilkan rancangan testbed SDN yang menggunakan perangkat *Raspberry Pi* sebagai *OpenVswitch* dan kontroler dengan topologi yang sama seperti pada simulator.

1.4 Lingkup

Adapun ruang lingkup dari Tugas Akhir ini adalah:

1. Melakukan implementasi dan pengujian *Multipath routing* di platform *SDN* pada simulator Mininet.
2. Merancang testbed SDN dengan menggunakan perangkat *Raspberry Pi* .

1.5 Pendekatan

1. Perumusan masalah

Pada tahapan ini didapatkan permasalahan pada *singlepath routing* dimana *singlepath routing* bekerja dengan cara memilih satu jalur dalam pengiriman paket dari *source* ke *destination* yang dapat menyebabkan terjadinya kemacetan yang berdampak kepada pemborosan sumber daya. Untuk mengatasi masalah ini maka digunakan metode *Multipath routing* yang bekerja dengan cara menggunakan banyak jalur pengiriman paket sehingga dapat meningkatkan *performance* jaringan. Dalam membuktikan kelebihan dari *Multipath routing* ini maka dilakukan *testbed* menggunakan *Raspberry Pi* sebagai *Open vSwitch*.

2. Tinjauan Pustaka

Pada tahapan ini dilakukan pencarian informasi yang terkait dengan materi tugas akhir seperti materi mengenai *singlepath routing*, *Multipath routing*, *Software Defined Network (SDN)* dan *Raspberry Pi* sebagai pengganti *Open vSwitch* yang

bersumber dari buku, media, jurnal dan diskusi yang bertujuan menunjang selesainya Tugas Akhir ini.

3. *Design dan analysis*

Pada tahap ini dilakukan perancangan penelitian Tugas Akhir untuk mengimplementasikan *Multipath routing* pada konsep jaringan *Software Define Network* (SDN) dan implementasi *Raspberry Pi* sebagai *Open vSwitch* sesuai dengan parameter yang dibutuhkan.

4. Implementasi dan Pengujian

Pada tahapan ini dijelaskan rincian implementasi *Multipath routing* di *Software Define Network* (SDN) dimana digunakan perangkat *raspberry pi* sebagai *Open vSwitch* untuk dapat melakukan *testbed*. Pada tahapan ini juga dijelaskan bahwa implementasi *Multipath routing* di *Software Define Network* (SDN) akan diuji dan akan dilakukan analisis untuk mendapatkan bukti bahwa implementasi *Multipath routing* di SDN dapat meningkatkan *performance* jaringan.

5. Kesimpulan dan saran

Pada tahapan ini didapat kesimpulan berdasarkan data pengujian pada tahapan yang telah dilakukan sebelumnya bahwa implementasi *Multipath routing* di *Software Define Network* (SDN) menggunakan *Raspberry Pi* sebagai *Open vSwitch* dapat meningkatkan *performance* jaringan.

1.6 Sistematika Penyajian

Secara garis besar dokumen ini disajikan dalam lima bab, dimana masing-masing bab pada tugas akhir ini disusun dalam sistematika berikut :

1. Bab I Pendahuluan

Pada bab ini berisi penjelasan mengenai latar belakang, tujuan pelaksanaan, ruang lingkup, pendekatan yang dilakukan, sistematika penyajian tugas akhir, serta istilah dan definisi yang digunakan dalam tugas akhir.

2. Bab II Tinjauan Pustaka

Pada bab ini menguraikan setiap dasar teori serta perangkat yang akan digunakan pada pengerjaan tugas akhir.

3. Bab III Analisis dan Perancangan Sistem

Pada bab ini akan dijelaskan mengenai analisis dan perancangan sistem yang akan diimplementasikan pada pengerjaan tugas akhir.

4. Bab IV Implementasi dan Pengujian

Pada bab ini berisi tentang hasil yang didapat setelah melakukan implementasi dan pengujian.

5. Bab VI Kesimpulan dan Saran

Pada bab ini dijelaskan mengenai kesimpulan dari pengerjaan tugas akhir serta saran yang diperlukan untuk pengembangan produk dimasa mendatang.

1.7 Istilah dan Definisi

Daftar istilah, definisi, dan singkatan yang terdapat pada dokumen ini dibuat untuk mempermudah pembaca dalam memahami setiap informasi yang terdapat dalam dokumen. Daftar istilah dan definisi yang terdapat dalam dokumen tugas akhir ini dapat dilihat pada Tabel 1 dan Tabel 2 berikut.

Tabel 1. Istilah

No	Istilah	Defenisi
1	SDN	<i>Software Define Network</i> adalah sebuah arsitektur jaringan yang memisahkan <i>control plane</i> dengan <i>data plane</i> untuk meningkatkan performa jaringan
2	<i>Singlepath</i>	Jalur tunggal yang ada pada arsitektur jaringan
3	<i>Routing</i>	menentukan jalur pada jaringan optimal yang akan ditempuh antar <i>host</i>
4	<i>Multipath</i>	memiliki lebih dari satu jalur pada pengiriman data antar <i>host</i>
5	Raspberry Pi	Komputer papan tunggal (<i>single-board circuit; SBC</i>) yang seukuran dengan kartu kredit yang dapat digunakan untuk menjalankan program perkantoran, permainan komputer, dan sebagai pemutar media hingga video beresolusi tinggi.
6	Trafik	Banyak nya lalu lintas pada jalur yang sama dalam jaringan
7	Bandwidth	Lebar jalur pada proses pengiriman paket atau data antar <i>host</i>

Daftar singkatan yang terdapat dalam dokumen tugas akhir ini dapat dilihat pada Tabel 2 berikut.

Tabel 2. Daftar Singkatan

No	Singkatan	Deskripsi
1.	SDN	<i>Software Define Network</i>
2.	TA	Tugas Akhir
3.	QoS	<i>Quality of Service</i>
4	ECMP	<i>Equal Cost Multipath Routing</i>

BAB II TINJAUAN PUSTAKA

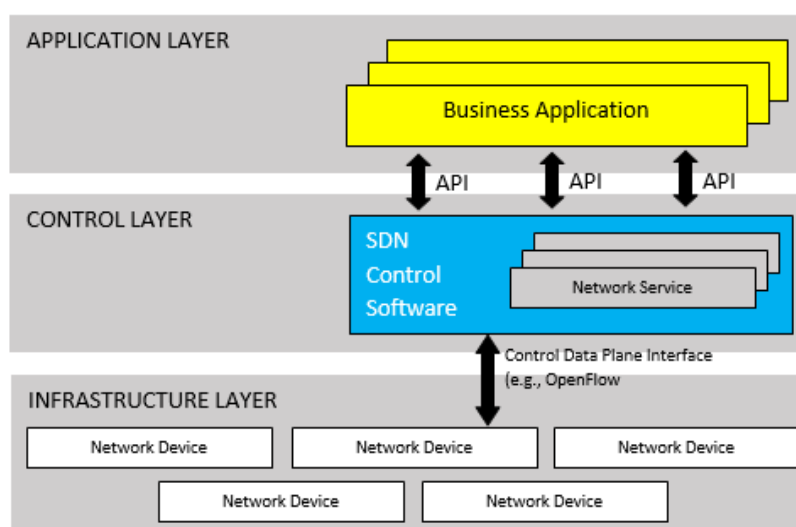
Pada bab ini akan dijelaskan landasan teori yang digunakan sebagai dasar teori dalam pengerjaan Tugas Akhir. Pada akhir bab ini akan dipaparkan penelitian terdahulu yang berhubungan dengan penelitian yang dilakukan pada Tugas Akhir ini.

2.1 Landasan Teori

Pada bagian landasan teori ini akan diuraikan secara ringkas mengenai komponen – komponen utama dan teknik yang digunakan dalam *implementation Multipath routing in SDN testbed*.

2.1.1 Software Define Network

Software-defined networking (SDN) adalah sebuah konsep pendekatan jaringan komputer dimana sistem pengontrol (*control plane*) dari arus data dipisahkan dari perangkat kerasnya (*data plane*). Umumnya, sistem pembuat keputusan kemana arus data dikirimkan dibuat menyatu dengan perangkat kerasnya. *Software-Defined Network* (SDN) adalah satu jaringan komputer yang sangat fleksibel karena SDN dikonfigurasi dan dikendalikan melalui *software* terpusat [5]. SDN ini di kembangkan oleh *Stanford University* yang mengeluarkan teknologi *Openflow*. Dengan pengaplikasian *Openflow*, pengguna dapat menganalisa kebutuhan gambaran pertumbuhan jaringan, yang dapat diamati langsung hanya melalui sebuah aplikasi. Melalui SDN tidak perlu bergantung pada vendor atau produk tertentu di dalam implementasi jaringan. SDN juga dapat digunakan untuk menciptakan sebuah jaringan universal [6] .



Gambar 1. Arsitektur SDN

Gambar 1 adalah SDN Arsitektur, menggambarkan bagaimana gambaran logikal dari SDN arsitektur, yang merupakan jaringan pintar yang tersentralisasi secara logis berdasarkan software (*software-based*). Selain itu ONF (Open Network Foundation) menyatakan bahwa dengan SDN tidak lagi membutuhkan protokol standar, tetapi cukup hanya menerima instruksi dari sebuah SDN kontroler [7].

Software Defined Networking adalah arsitektur jaringan yang bersifat *dynamic*, *manageable*, *cost-effective*, dan *adaptable*. Arsitektur SDN bertujuan untuk membuat jaringan menjadi lebih fleksibel dan mempermudah dalam mengontrol jaringan apabila

terdapat perubahan dalam business requirement. Pada SDN, *network administrator* atau *network engineer* dapat membentuk lalu lintas jaringan melalui sebuah *central console*, sehingga tidak perlu mengkonfigurasi masing-masing *switch* atau perangkat yang terdapat pada topologi. SDN juga memungkinkan *administrator* sistem untuk mempercepat koneksi penyediaan jaringan.

2.1.2 *Singlepath Routing*

Singlepath adalah mempelajari jalur yang terbaik dan hanya memilih satu destinasi saja di dalam penggunaan *singlepath* mempunyai kelemahan dalam penggunaan jalurnya yaitu penggunaan jalur tunggal yang dapat menyebabkan terjadinya kemacetan (*bottleneck*) dan tabrakan antar paket (*congestion*) jika trafik tinggi. Kondisi ini berdampak pada *throughput* jaringan yang menurun bahkan paket yang dikirimkan dapat di-drop. Kelemahan lainnya, *singlepath* tidak dapat memfasilitasi trafik data dari aplikasi tertentu yang membutuhkan prioritas *Quality of Service* (QoS) yang handal [1].

2.1.3 *Multipath Routing*

Multipath Routing adalah teknik *routing* menggunakan beberapa jalur alternatif melalui jaringan, yang dapat menghasilkan berbagai keuntungan seperti *fault tolerance*, peningkatan *bandwidth*, dan peningkatan keamanan jaringan. Peningkatan performa jaringan dengan *Multipath routing* dicapai melalui penggunaan beberapa jalur secara bersama selama proses transmisi data berlangsung. *Multipath routing* juga menggunakan metode *congestion control* untuk menentukan seberapa banyak trafik yang dilewatkan pada setiap jalur. Pendekatan ini membuat penggunaan sumber daya jauh lebih efektif dan meningkatkan *throughput* jaringan [8].

2.1.3.1 ECMP

Equal Cost Multipath Routing (ECMP) adalah proses *routing* yang memanfaatkan seluruh pilihan jalur yang tersedia antara satu *node* dengan *node* lainnya dalam jaringan dengan memetakan setiap jalur dengan seluruh kemungkinan trafik (pasangan *source-destination*) secara statis. Konfigurasi tersebut memungkinkan adanya pengiriman dua atau lebih trafik melalui rute-rute dengan komponen link yang beririsan, meskipun ada rute alternatif yang sedang tidak digunakan. *Software Defined networking* (SDN) memiliki kemampuan untuk membuat ECMP lebih dinamis dengan cara mengukur *bandwidth* yang tersisa dari setiap link dalam jaringan secara *real-time* oleh *Controller*. Hasil pengukuran *bandwidth* yang dilakukan kemudian dijadikan dasar oleh *Controller* untuk melakukan penentuan jalur yang dilewati sebuah trafik [9].

ECMP memungkinkan penggunaan beberapa jalur yang memiliki nilai *cost* yang sama dalam perutean. Fitur ini bukan hanya akan membantu mendistribusikan lalu lintas lebih merata, tetapi juga sebagai metode proteksi atau keamanan dalam jaringan. Dengan menggunakan ECMP, jalur yang memiliki nilai yang sama akan dipasang ke dalam *table load balancing* di lapisan *forwarding* pada *router*. Pada saat *router* mendeteksi adanya kegagalan link, trafik akan didistribusikan secara otomatis kepada jalur kedua dengan tanpa adanya kehilangan trafik yang cukup signifikan. ECMP tidak memerlukan konfigurasi secara khusus, sebab algoritma SPF akan menghitung secara otomatis jalur-jalur dengan nilai *cost metric* yang sama, untuk kemudian melakukan *advertising* kepada layer *forwarding* pada *router*. Jumlah jalur ECMP bergantung kepada *algoritma load balancing* yang mendukung. Umumnya, dikonfigurasi antara 1 sampai 8 dan 16 jalur, bergantung

kepada nilai maksimum dari jalur yang ada, cara kerja algoritma ECMP dapat dilihat pada flowchart dibawah ini [10].

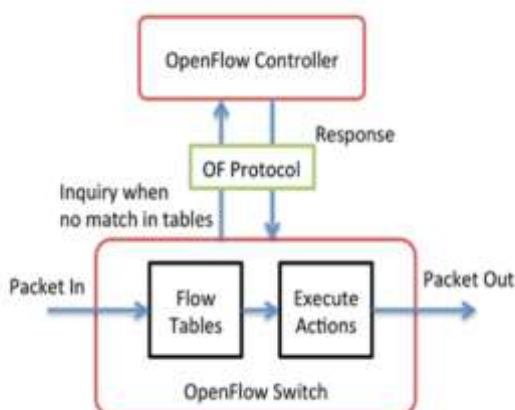
2.1.3.2 Open Shortest Path First(OSPF)

OSPF merupakan salah satu protokol *routing* berbasis teknologi *Shortest Path First* (SPF) dan *link state*. Masing – masing router memiliki *link-state* database yang identik, yang mendeskripsikan topologi kondisi jaringan pada sebuah, yaitu informasi mengenai topologi *Autonomous System* (AS), *interface* yang tersedia, *neighbor router*, dan informasi lain berkaitan dengan jalur pendistribusian data. OSPF merupakan protokol *routing* yang menggunakan konsep hirarki *routing*, artinya OSPF membagi-bagi jaringan menjadi beberapa tingkatan. Tingkatan-tingkatan ini diwujudkan dengan menggunakan sistem pengelompokan area. Konsep ini menjadikan sistem penyebaran informasinya menjadi lebih teratur dan tersegmentasi. Efek dari keteraturan distribusi *routing* ini adalah jaringan yang penggunaan bandwidth-nya lebih efisien, lebih cepat mencapai konvergensi, dan lebih presisi dalam menentukan rute terbaik menuju ke sebuah lokasi. Hal ini membuat protokol *routing* OSPF menjadi sangat cocok untuk terus dikembangkan menjadi jaringan berskala besar [11].

2.1.4 Openflow

Openflow adalah protokol terbilang relatif baru yang dirancang dan diimplementasikan di *Stanford University* pada tahun 2008. Protokol *Openflow* telah berkembang sejak dimulainya proses standarisasi oleh *Open Network Foundation* (ONF), yaitu sebuah organisasi yang berfokus dalam pengembangan protokol *Openflow*, dari versi 1.0–1.5. *Openflow* terus berkembang dengan menambahkan fitur- fitur baru, dibawah ini adalah *Roadmap* perkembangan *Openflow* dari versi 1.0–1.5. *Openflow* adalah salah satu jenis dari APIs (*Application Protocol Interfaces*) dalam jaringan SDN yang digunakan untuk mengontrol/mengatur *traffic flows* pada *switch* dalam sebuah jaringan, jadi *control plane* berkomunikasi dengan *data plane* melalui *Openflow* [12].

Openflow mempunyai 2 komponen penting yaitu *Openflow Controller* dan *Openflow Switch* yang dapat dilihat pada gambar dibawah ini.



Gambar 2.Sistem Kerja Openflow

(source: <http://www.fiber-optic-transceiver-module.com/openvswitch-vs-openflow-what-are-they-whats-their-relationship.html>)

1. *Openflow Controller*

Controller bertugas mengontrol *path*, memformulasikan *flow* dan mengatur kerja dari *Openflow switch*. Terdapat beberapa *Openflow Controller* yang dapat digunakan seperti *Ryu* (Phyton base), *NOX*(C base), *POX*(python base), dan *Floodlight* (java base) .

2. *Openflow Switch*

Implementasi *Openflow Switch* dapat dilakukan dengan berbagai cara seperti :

- menggunakan *hardware Openflow switch*
- menggunakan *OpenVswitch* yang diinstall di sistem operasi berbasis linux
- menggunakan *ethernet card w/ NetFPGA support*
- menggunakan mininet *Openflow switch emulator*

2.1.4.1 OpenVSwicth

OpenVswitch adalah sebuah *switch* virtual berbasis linux. *OpenVswitch* sendiri merupakan perangkat lunak sumber terbuka yang berada dalam lisensi Apache 2.0. *OpenVswitch* mendukung pengontrolan secara otomatis dengan protokol *openflow*. *OpenVswitch* dirancang untuk memungkinkan otomatisasi jaringan yang besar melalui ekstensi terprogram, sambil tetap mendukung antarmuka dan protokol manajemen standar seperti *NetFlow*, *sFlow*, *IPFIX*, *RSPAN*, *CLI*, *LACP*, dan *802.1ag* [13].

2.1.4.2 Raspberry Pi

Raspberry Pi yang disingkat dengan nama *Raspi* merupakan suatu perangkat keras yang berukuran kecil layaknya kartu kredit yang dapat digunakan untuk menjalankan berbagai program yang diinginkan pengguna. *Raspberry Pi* yang dapat digunakan untuk menjalankan program perkantoran, permainan komputer, dan sebagai pemutar media hingga video beresolusi tinggi. Beberapa fungsi *Raspberry Pi* adalah sebagai Komputer Desktop Mini, sebagai *File Server*, sebagai *Download Server*, sebagai *Access Point*, sebagai *Server DNS*, sebagai *Multimedia Player*



Gambar 3. Raspberry Pi

Raspberry Pi memiliki dua model: model A dan model B. Secara umum *Raspberry Pi* Model B memiliki kapasitas penyimpanan RAM sebesar 512 MB. Perbedaan model A dan B terletak pada modul penyimpanan yang digunakan. Model A menggunakan penyimpanan sebesar 256 MB dan penyimpanan model B sebesar 512 MB. Selain itu, model B sudah dilengkapi dengan *port Ethernet* (untuk LAN) yang tidak terdapat di model A. Desain *Raspberry Pi* didasarkan pada SoC (system-on-a-chip) Broadcom BCM2835, yang telah menanamkan prosesor ARM1176JZF-S dengan 700 MHz, GPU VideoCore IV, dan RAM sebesar 256 MB (model B). Penyimpanan data tidak didesain untuk menggunakan cakram keras atau *solid-state drive*, melainkan mengandalkan kartu penyimpanan tipe SD untuk menjalankan sistem dan sebagai media penyimpanan jangka panjang. Pada tugas akhir ini *Raspberry pi* berfungsi sebagai *Controller* dan sebagai *OpenVswitch* sebagai bentuk implementasi *Multipath routing in SDN Testbed* [14].

2.1.5 Ryu

Ryu merupakan salah satu *Controller* dalam *software defined network* yang dirancang untuk meningkatkan kemampuan *network* dengan membuat konfigurasi jaringan lebih mudah untuk dikelola [9]. Ryu menggunakan *application program interface* (API) yang sudah didefinisikan dengan sangat baik sehingga dapat melakukan pengembangan dengan mudah untuk membuat suatu *network management* yang baru [10]. Ryu merupakan *Controller* yang menggunakan bahasa pemrograman *python* yang mudah dalam pemakaiannya serta memiliki dokumentasi yang banyak sehingga lebih mudah menemukan solusi jika terdapat permasalahan. *Controller Ryu* ini mendukung beberapa protokol dalam *software defined network* diantaranya *Openflow*, *Netconf*, *OF-config* serta lainnya. *Framework Ryu* berada pada *Control Layer* pada Arsitektur SDN, dan beberapa aplikasi pada Ryu berada pada *Application Layer* guna untuk berkomunikasi dengan Aplikasi SDN lain menggunakan API seperti REST, RPC, dan sebagainya [15].

Jika menggunakan protokol *Openflow*, *Controller Ryu* memiliki beberapa fungsi utama seperti:

1. *Packet-In Handler* : Digunakan untuk menerima *packet-in* dari setiap *switch*, untuk kemudian di ekstrak datanya (contoh: *ip address* asal dan *ip address* tujuan) dan dilakukan kalkulasi yang dibutuhkan.
2. *Datapath.send_msg*: Digunakan untuk mengirimkan instruksi terhadap *switch* melalui protokol *Openflow*.
3. *Event SwitchEnter* dan *EventLinkAdd*: Digunakan untuk mengetahui identitas *switch* dan pemetaan *link* antar *switch* pada saat *switch* pertama kali terhubung dengan *Controller*.
4. *Traffic Stats Reply Handler*: Digunakan untuk menampung laporan kondisi jaringan setelah mengirim *Traffic Stats Request*.

2.1.6 ICMP (Internet Control Message Protocol)

ICMP (*Internet Control Message Protocol*) merupakan bagian dari *Internet Protocol*. Pesan ICMP dikirimkan dalam paket IP, dan digunakan untuk mengirim pemberitahuan yang berhubungan dengan kondisi jaringan. Beberapa fungsi utama ICMP adalah sebagai berikut:

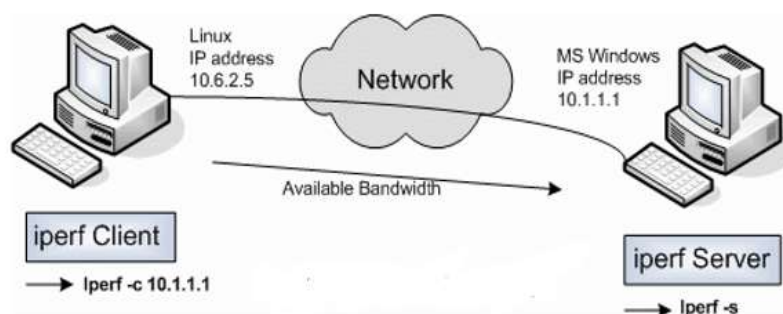
1. Memberitahu *error* (kesalahan) jaringan, seperti seluruh jaringan atau *host unreachable* karena beberapa jenis kegagalan. Termasuk paket TCP atau UDP yang diarahkan ke nomor *port* tertentu yang tidak ada penerima yang menerimanya.
2. Memberitahu kemacetan jaringan. Ketika banyak paket ditahan oleh *router* karena ketidakmampuan untuk mengirimkan paket tersebut secepat paketnya diterima, *router* akan mengirimkan pesan ICMP *Source Quench*. Tetapi mengirimkan terlalu banyak *Source Quench* akan menyebabkan jaringan menjadi lebih macet.
3. Membantu *Troubleshooting*. ICMP mendukung fungsi Echo, yang mana paket hanya dikirimkan dalam rentetan antara dua *host*. Ping merupakan salah satu tool manajemen jaringan yang berbasis pada fitur ini. Ping akan mengirimkan rentetan paket, mengukur waktu perjalanan rata-rata dan menghitung persentase kerugiannya.
4. Memberitahu *Timeout*. Jika ada bagian TTL paket IP yang di-drop ke nol (0), *router* yang membuang paket akan mengirimkan paket pesan ICMP untuk memberitahu hal tersebut. *Traceroute* merupakan tool yang memetakan jaringan dengan mengirim paket-paket dengan nilai TTL kecil dan melihat pemberitahuan ICMP timeout.

2.1.7 Iperf

Iperf dikembangkan oleh NLANR (National Laboratory for Applied Network Research) / DAST (Distributed Applications Support Team) sebagai alternatif *modern* untuk melakukan pengujian *performance* jaringan berupa pengukuran *bandwidth* TCP dan pengukuran kinerja UDP secara maksimal. Iperf memungkinkan tuning berbagai parameter dan karakteristik UDP dan TCP. Iperf melaporkan hasil *bandwidth*, *delay*, dan paket *loss* datagram disetiap hasil pengukurannya .

Iperf dapat mengukur *bandwidth* diukur melalui tes TCP. Untuk lebih jelas, perbedaan antara TCP (*Transmission Control Protocol*) dan UDP (*User Datagram Protocol*) adalah bahwa TCP menggunakan proses untuk memeriksa bahwa paket-paket itu dikirim dengan benar ke penerima sedangkan dengan UDP paket-paket dikirim tanpa pemeriksaan tetapi dengan keuntungan menjadi lebih cepat dari TCP. Iperf menggunakan perbedaan kapasitas TCP dan UDP untuk menghasilkan statistik tentang *network links*. Iperf dapat diinstal dengan sangat mudah pada sistem UNIX / Linux atau Microsoft Windows [17].

Berikut adalah diagram di mana Iperf diinstal pada mesin Linux dan Microsoft Windows.



Gambar 4. Diagram Iperf
(source: <https://openmaniak.com/iperf.php>)

2.1.8 Mininet

Mininet adalah sebuah *network emulator* untuk mendukung pengembangan SDN. Mininet dapat menjalankan beberapa perangkat seperti *end-host*, *switch*, *router*, dan *link* yang mengkoneksi perangkat jaringan tersebut diatas *Linux Kernel*. Mininet menggunakan konsep virtualisasi yang bisa membuat perangkat-perangkat jaringan secara virtual dan mampu mengkoneksikan perangkat-perangkat jaringan tersebut menjadi suatu topologi jaringan yang utuh [18].

Mininet memungkinkan pengembangan SDN pada laptop atau PC apa pun, dan desain SDN dapat dijalankan dengan baik pada Mininet (memungkinkan pengembangan yang murah dan efisien), dan perangkat keras yang nyata berjalan dengan kecepatan line dalam penyebaran langsung. Mininet memungkinkan pembuatan prototipe SDN, pengujian topologi yang kompleks tanpa perlu memasang jaringan fisik dan bekerja mandiri antara *Multiple concurrent developers* pada topologi yang sama.

Jaringan Mininet menjalankan kode nyata termasuk aplikasi jaringan Unix / Linux standar serta kernel Linux asli dan *network stack*. Mininet menyediakan API Python yang dapat dikembangkan untuk pembuatan dan eksperimen jaringan. Ini dirilis di bawah lisensi *Open Source BSD* yang permisif dan secara aktif dikembangkan dan didukung oleh komunitas penggemar jaringan dan SDN. Jaringan mininet terdiri atas:

1. *Isolated Hosts*

Sebuah group dari *user level processes* yang berpindah ke *network namespace* yang memberikan kepemilikan eksklusif antarmuka, port, dan tabel perutean.

2. *Emulated LinksLinux*

Traffic Control (tc) memberlakukan *data rate* setiap tautan untuk membentuk lalu lintas ke *rate* yang dikonfigurasi. Setiap *host* yang ditiru memiliki antarmuka *Ethernet* virtualnya sendiri.

3. *Emulated Switches*

Linux Bridge default atau *Open vSwitch* yang berjalan dalam mode kernel digunakan untuk mengalihkan paket antar antarmuka. *Switch* dan *router* dapat berjalan di kernel atau di ruang pengguna.

2.1.9 Metrik

Untuk dapat melihat peningkatan performa jaringan maka akan dilakukan beberapa pengukuran dengan metrik-metrik seperti *throughput*, *bandwidth*, dan *delay*.

2.1.9.1 Throughput

Throughput adalah kecepatan rata-rata data yang diterima oleh suatu *node* dalam selang waktu pengamatan tertentu. Data yang dimiliki oleh pesan-pesan ini dapat dikirimkan melalui tautan fisik atau logis, atau dapat melewati simpul jaringan tertentu. *Throughput* biasanya diukur dalam bit per detik (bit/s atau bps), dan kadang-kadang dalam paket data per detik (p/s atau pps) atau paket data per slot waktu [19].

Throughput sistem atau *throughput* agregat adalah jumlah dari laju data yang dikirimkan ke semua terminal dalam suatu jaringan. *Throughput* pada dasarnya identik dengan konsumsi *bandwidth digital*, itu dapat dianalisis secara matematis dengan menerapkan teori

antrian, di mana beban dalam paket per unit waktu dilambangkan sebagai tingkat kedatangan (λ), dan *throughput*, di mana penurunan paket per unit waktu, dilambangkan sebagai tingkat keberangkatan (μ).

Throughput sistem komunikasi dapat dipengaruhi oleh berbagai faktor, termasuk keterbatasan media fisik analog yang mendasarinya, kekuatan pemrosesan yang tersedia dari komponen sistem, dan perilaku pengguna akhir. Ketika berbagai *overhead* protokol diperhitungkan, tingkat yang berguna dari data yang ditransfer dapat secara signifikan lebih rendah dari *throughput* maksimum yang dapat dicapai.

2.1.9.2 Bandwidth

Bandwidth adalah besaran yang menunjukkan seberapa banyak data yang dapat dilewatkan dalam koneksi melalui sebuah *network*. *Bandwidth* biasanya dihitung bit per detik, atau Megabits per detik, dinyatakan sebagai kbit/s atau Mbit/s. *Bandwidth* diukur kotor, jumlah data yang ditransfer dalam periode waktu tertentu dinyatakan dalam tingkat, tanpa mempertimbangkan kualitas dari sinyal itu sendiri [19]. Terdapat dua pembagian *bandwidth*, yakni *bandwidth* analog dan *bandwidth* digital:

1. *Bandwidth analog* merupakan jenis *bandwidth* yang digunakan untuk menyatakan adanya perbedaan frekuensi rendah dengan frekuensi tinggi. Perbedaan tersebut diukur dalam rentang waktu dan dengan satuan tertentu yakni Hertz atau Hz. Dari sini bisa memperoleh informasi seputar berapa banyak informasi yang dikirimkan dalam waktu tertentu. *Bandwidth* analog memang jarang digunakan dalam jaringan. Namun bukan berarti *bandwidth* analog ini tidak penting karena dari sini bisa memperoleh informasi penting mengenai berapa banyak informasi yang sudah disampaikan ke alamat tujuan.
2. *Bandwidth Digital* merupakan jenis *bandwidth* yang digunakan untuk mengetahui jumlah data yang bisa untuk dikirimkan, baik ketika menggunakan kabel atau nirkabel atau *wireless*. Satuan untuk menghitung *bandwidth* digital adalah *bit*. *Bandwidth* digital sering digunakan oleh para penyedia internet atau ISP yang ada di Indonesia.

2.1.9.3 Delay

Delay adalah waktu tunda yang disebabkan oleh proses transmisi dari satu titik ke titik lain yang menjadi tujuannya. *Delay* dalam jaringan dapat digolongkan sebagai berikut :

1. *Packetization Delay*

Delay yang disebabkan oleh waktu yang diperlukan untuk proses pembentukan paket IP dari informasi *user*. *Delay* ini hanya terjadi sekali, yaitu di *source* informasi.

2. *Queuing Delay*

Delay ini disebabkan oleh waktu proses yang diperlukan oleh *router* di dalam menangani antrian transmisi paket di sepanjang jaringan. Umumnya *delay* ini sangat kecil, kurang lebih 100 *microsecond*.

3. *Delay Propagasi*

Proses perjalanan informasi selama didalam media transmisi, misalnya SDH, coax atau tembaga, menyebabkan *delay* yang disebut dengan *delay* propagasi.

4. *Transmission Delay*

Transmission delay adalah waktu yang diperlukan sebuah paket data untuk melintasi suatu media. *Transmission delay* ditentukan oleh kecepatan media dan besar paket data.

5. *Processing delay*

Processing delay adalah waktu yang diperlukan oleh suatu perangkat jaringan untuk melihat rute, mengubah *header*, dan tugas *switching* lainnya

2.2 Related Work

Pada subbab ini akan dibahas mengenai jurnal atau penelitian terkait dengan sistem yang akan dibangun, berikut adalah jurnal yang terkait:

2.2.1 SDN Implementation of *Multipath* Discovery to Improve Network Performance in Distributed Storage Systems

Jurnal ini membahas tentang penggunaan dari *Distributed Storage Systems* (DSS) dapat meningkatkan kebutuhan untuk transfer data yang efektif dari *storage* ke *storage*. Sebagian besar solusi perutean masih menggunakan jalur tunggal. Dalam makalah ini, menyajikan pendekatan pragmatis untuk *routing Multipath* di DSS, yang didasarkan pada *Software Defined Networking*. SDN digunakan karena memungkinkan peningkatan jaringan programmability kerja dengan memisahkan bidang kontrol dari *data (forwarding) plane*, yang membuatnya ideal untuk administrasi lalu lintas dalam skenario yang dinamis dan efisien perhitungan menentukan kinerja keseluruhan. Penemuan jalur dihitung dengan menemukan jalur disjoint k-maksimum dalam multigraf. Hasil awal menunjukkan bahwa, dengan menggunakan solusi *Multipath*, tidak hanya peningkatan throughput keseluruhan tetapi juga efisiensi penggunaan sumber daya [16].

2.2.2 HiQoS: An SDN-Based *Multipath* QoS Solution

Jurnal ini membahas tentang implementasi dan pengontrol secara berkala mengukur paket yang dikirimkan dari setiap antrian di jalur, dan menghitung jalur optimal. Kemudian *Controller* mendorong *flow entri* ke *Openflow switches*. jurnal juga menetapkan tanda yang mewakili kondisi pengiriman data. *Controller* menghapus entri aliran dan daftar *cache Multipath* setelah beberapa waktu pengiriman data. Menggunakan QoS banyak klien sedang mengakses *server 1* dan *server 2*. QoS meningkatkan lalu lintas melebihi *bandwidth* jaringan, sehingga dapat mengukur *throughput* yang baik dari masing-masing solusi QoS. percobaan dalam jurnal secara berkala mengukur throughput dari 1 sampai 5 detik untuk menghitung kalkulasi throughput rata-rata system[22].

2.2.3 *Multipath Routing* in SDN for Network Performance Improvement

Jurnal ini membahas tentang *Multipath routing* yang dapat digunakan untuk berbagai tujuan, termasuk *balancing* beban lalu lintas, agregasi *bandwidth* untuk meningkatkan *throughput*, menyediakan jaminan menyeluruh dan meningkatkan toleransi kesalahan jaringan dan *throughput*. Tujuan utama dari *Multipath routing* di SDN untuk meningkatkan *throughput* serta pemanfaatan sumber daya dari sistem. *Throughput* sistem dapat ditingkatkan dengan *paket loss* yang rendah dan *packet-loss* persentase di bawah uji TCP. Terutama itu menunjukkan bahwa ketika jalur tunggal digunakan, persentase kerugian sangat tinggi hingga 90%, jadi itu adalah pilihan yang baik untuk menerapkan teknik seperti *Multipath* SDN dalam sistem penyimpanan terdistribusi [23].

BAB III ANALISIS DAN PERANCANGAN

Bab ini menguraikan bagaimana proses pelaksanaan tugas akhir dilakukan dalam menyelesaikan persoalan yang dipertanyakan. Bab ini dapat dipecah menjadi beberapa sub bab jika diperlukan.

Di bab ini tidak disampaikan hasil pelaksanaan kajian, tetapi apa saja yang dilakukan dan bagaimana melakukannya. Sekali lagi, tidak mengemukakan hasil pelaksanaan kajian.

3.1. Analisis

Pada sub bab ini dibahas mengenai analisis masalah, analisis pemecahan masalah dan analisis kebutuhan sistem untuk menentukan solusi pemecahan terhadap masalah yang terjadi.

3.1.1 Analisis Masalah

Jaringan internet telah mengubah pola hidup manusia dalam bertukar informasi sehingga menggiring adanya revolusi komunikasi secara signifikan. Pertumbuhan jumlah pengguna dan layanan pada jaringan berdampak kepada peningkatan kualitas *provider* jaringan untuk memenuhi permintaan pengguna. Untuk merespon hal ini, diperlukan adanya optimasi kinerja jaringan melalui adanya rekayasa lalu lintas dengan memanfaatkan sumber daya jaringan secara efisien dan handal.

Routing adalah suatu proses untuk memindahkan suatu paket data dalam sebuah jaringan komputer dari *node* asal sampai dengan *node* tujuan. Tujuan dari *routing* adalah untuk menemukan route terbaik untuk mengirimkan atau mentransmisikan paket data. Algoritma dan teknik *routing* terus di kembangkan untuk mendapatkan hasil paling optimal dan mengurangi beban jaringan pada saat melakukan transmisi data. Sebagian besar protokol *routing* yang digunakan saat ini menggunakan algoritma yang menghasilkan jalur tunggal saja. Perhitungan ini dilakukan secara otomatis dengan menggunakan nilai satuan beban (metrik) yang dihitung berdasarkan jarak, *throughput*, *delay*, *bandwidth*, atau kombinasi keduanya. Selain itu, nilai metrik ini ditetapkan tanpa mempertimbangkan permintaan trafik atau beban trafik pada jaringan .

Kemacetan dalam pengiriman data dari *host source* ke *host destination* terjadi ketika jumlah paket yang ditransmisikan melalui jaringan telah mendekati kapasitas penanganan paket yang dapat dilakukan. Sedangkan penggunaan jalur tunggal dalam proses transmisi data pada siklus trafik yang cenderung tinggi akan menyebabkan terjadinya kemacetan (*bottleneck*) dan tabrakan antar paket (*congestion*) yang berdampak kepada pemborosan sumber daya jaringan.

3.1.2 Analisis Pemecahan Masalah

Pemecahan masalah yang dilakukan adalah dengan mengimplementasikan *Software Defined Network* (SDN). SDN merupakan arsitektur jaringan yang memiliki konsep dimana sistem *control plane* dan *forwarding/data plane* dipisahkan. Konsep SDN ini berbeda Pada jaringan tradisional dimana *control plane* dan *forwarding/data plane* tidak terpisah melainkan menjadi satu dalam sebuah perangkat jaringan yang sama. Tujuan dari konsep SDN ini agar jaringan dapat diprogram secara langsung dan manajemen jaringan menjadi lebih baik dari jaringan tradisional sebelumnya. *Control plane* pada SDN bertindak sebagai

komponen pada jaringan yang mengontrol jaringan secara terpusat. Kemudian untuk *data plane* merupakan bagian yang bertugas untuk meneruskan paket, mengatur QoS, menguraikan header paket serta enkapsulasi paket. Oleh karena itu SDN memiliki arsitektur yang terpusat dan *programmable* sehingga sangat baik dalam pengelolaan suatu sistem jaringan [20].

Penelitian ini juga menggunakan teknik *multipath routing* dimana teknik *routing* ini menggunakan beberapa jalur alternatif dalam pengiriman paket melalui jaringan. Dengan mengimplementasikan *multipath routing* diharapkan dapat meningkatkan *throughput* pengiriman paket data sehingga dapat menunjang *performance* jaringan. Peningkatan performa jaringan dengan menggunakan *Multipath routing* dicapai melalui penggunaan beberapa jalur secara bersama selama proses transmisi data berlangsung. Dalam mengimplementasikan *multipath routing* ini ditambahkan algoritma yang akan melakukan perhitungan dan pengukuran dalam melakukan pengiriman paket serta mengatur sistem kerja dari flow yang akan dipakai untuk mengirimkan paket. ECMP merupakan algoritma yang dipakai dalam melakukan penelitian ini, algoritma ECMP digunakan untuk menguji apakah SDN dapat bekerja dengan konsep *load balancing* ketika *traffic* tinggi pada beberapa path yang akan dilalui oleh paket data [21].

Penggabungan antara teknik *Multipath routing* dan konsep SDN diharapkan dapat semakin meningkatkan *performance* jaringan dalam transmisi data dari *source* ke *destination*. *Multipath routing* dengan algoritma ECMP akan menggunakan beberapa jalur untuk mengirimkan paket data sedangkan SDN menggunakan konsep pemisahan kontrol *plane* dan *data plane* sehingga lebih dinamis, *programmable*, dan mudah dalam beradaptasi dengan vendor-vendor yang berbeda. Dengan demikian, penelitian ini diharapkan dapat meningkatkan performa jaringan dengan mengimplementasikan *multipath routing* pada SDN *testbed*.

3.1.3. Analisis Kebutuhan Sistem

Pada sub bab ini dijelaskan mengenai kebutuhan yang diperlukan selama proses pembangunan *Implementing Multipath Routing In SDN Testbed*. Kebutuhan tersebut mencakup kebutuhan perangkat keras dan perangkat lunak

3.1.3.1 Kebutuhan Perangkat Keras

Perangkat Keras yang dibutuhkan dalam membangun sistem pada Tugas Akhir ini dapat dilihat pada Tabel 3.

Tabel 3. Kebutuhan Perangkat Keras

No	Perangkat	Spesifikasi	Keterangan
1	PC/ Komputer	<i>Operating System : Windows 10Educ</i> <i>Processor : Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz</i> 3.30 GHz RAM : 8GB System type : 64-bit	Menulis Dokumen Tugas Akhir
2	Raspberry Pi 3 B	SOC: Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit	Sebagai <i>Controller</i> dan Open Vswitch

No	Perangkat	Spesifikasi	Keterangan
		SoC. CPU: 1.4GHz 64-bit quad-core ARM Cortex-A53 CPU. RAM: 1GB LPDDR2 SDRAM. WIFI: Dual-band 802.11ac wireless LAN (2.4GHz and 5GHz) and Bluetooth 4.2. Ethernet: Gigabit Ethernet over USB 2.0 (max 300 Mbps). Thermal management: Yes	
3	LAN/ Ethernet	Panjang : 2 Meter 20 buah	Sebagai penghubung jaringan antara <i>Controller</i> , <i>Open Vswitch</i> dan <i>Host</i>
4	USB to LAN Adapter	16 buah	Menghubungkan Lan/ <i>Ethernet</i> melalui <i>port</i> USB

3.1. 3.2 Kebutuhan Perangkat Lunak

Perangkat lunak yang dibutuhkan dalam membangun sistem pada Tugas Akhir ini dapat dilihat pada Tabel 4.

Tabel 4. Kebutuhan Perangkat Lunak

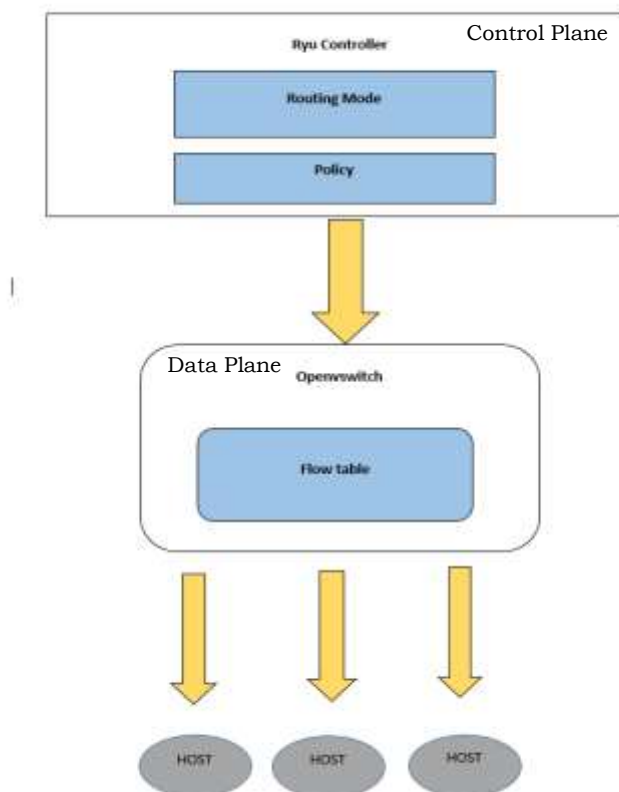
No	Software	Keterangan
1	Mininet	Sebagai simulator untuk pengujian jaringan SDN
2	NOOBS	<i>Operating System</i> pada raspberry pi 3
3	Thonny Phyton	<i>Editor</i> program Python pada raspberry pi 3
4	Google Docs	Sebagai tempat pengerjaan Tugas Akhir ini
5	POX	Digunakan sebagai kontroler pada SDN menggunakan mininet
6	Ryu	Digunakan sebagai SDN Framework

3.2 Perancangan Pembangunan Sistem

Perancangan merupakan tahap awal dalam *Implementing Multipath Routing In SDN Testbed* yang melibatkan *hardware* dan *software*. Perancangan dijadikan sebagai acuan untuk perakitan alat, pembuatan program dan implementasi *Implementing Multipath Routing In SDN Testbed*. Tujuan dari perancangan sistem ini adalah untuk mempermudah di dalam merealisasikan pembangunan sistem dan program yang sesuai dengan sumber daya yang dimiliki serta spesifikasi perangkat yang dibutuhkan. Oleh karena itu, dengan adanya tahap perancangan ini, segala kemungkinan yang dapat menghambat dan membatasi pengerjaan sistem ini dapat diminimalisir.

3.2.1 Sistem Arsitektur

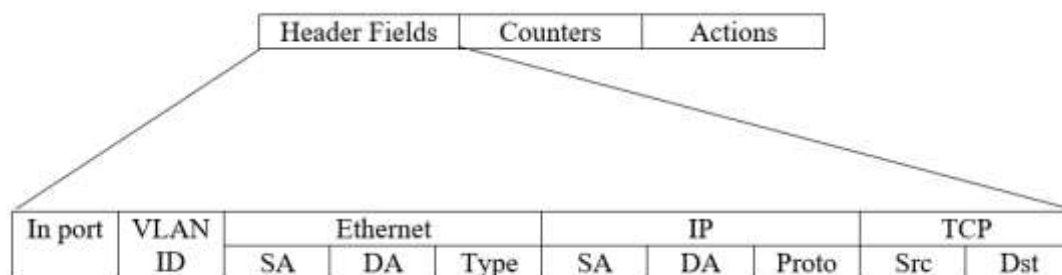
Dalam mengimplementasikan sistem yang akan dibangun, dibutuhkan pemahaman yang baik tentang sistem arsitekturnya. Dimana pada tugas akhir ini berfokus pada implementasi SDN dan *Multipath routing*. SDN menggunakan konsep pemisahan antara *data plane* dan *control plane*. Tujuannya agar dapat diprogram secara langsung dan manajemen jaringan menjadi lebih baik dari jaringan tradisional sebelumnya. Berikut gambaran arsitektur hubungan antara *control plane* dan *data plane*.



Gambar 5. Arsitektur Control Plane dan Data Plane

Pada SDN, control plane dan data plane dipisah dalam perangkat jaringan yang berbeda, oleh karena itu SDN memiliki arsitektur yang terpusat. *Controller* memiliki fungsi yaitu melakukan konfigurasi secara terpusat, pengelolaan jaringan, dan penentuan informasi *routing table* serta menerapkan *policy* pada *data plane*. Ada banyak *Controller* yang tersedia untuk membangun jaringan berbasis *Software Defined Network* diantaranya yaitu *OpenDaylight*, *Onos Floodlight*, *Beacon*, *Ryu*, *Pox*, dll. Dalam Tugas akhir ini berfokus pada penggunaan *Controller Ryu* untuk mengatur sistem SDN. Ryu menyediakan komponen perangkat lunak *Application Programming Interface* (API) yang bersifat *programmable* yang dapat memudahkan pengembang dalam membuat aplikasi manajemen dan kontrol jaringan baru, dengan menulis algoritma dan model fungsi. Ryu dapat mengontrol jaringan dengan bertukar informasi dengan *data plane* di jaringan. Di Dalam *data plane*, *OpenVswitch* digunakan sebagai *network devices* untuk meneruskan data dari satu *host* ke *host* lain. Setiap *OpenVswitch* yang saling terhubung memiliki group antrian didalam *port*.

Controller menggunakan protokol *Openflow* untuk melakukan konfigurasi perangkat. *Openflow* digunakan sebagai protokol komunikasi yang berada di lingkungan SDN untuk melakukan interaksi langsung dengan fungsi forwarding dari SDN *Controller* menuju SDN switch. *Openflow* berjalan pada koneksi Transmission Control Protocol (TCP) dan Transport Layer Security (TLS) sehingga *OpenVswitch* dapat memulai koneksi dan mengirim permintaan koneksi ke *Controller*. *Controller* terlibat langsung dalam memasang aturan pada saat paket masuk ke *data plane*. Dalam mengontrol *OpenVswitch* jaringan, *Controller* akan mengirim aturan ke *OpenVswitch*, aturan tersebut biasa disebut *flow rule* yang disimpan dalam *flow table*. Ketika paket datang ke *OpenVswitch*, jika tidak ada *flow rule* pada *flow table* yang cocok, maka paket tersebut dikirim ke *Controller*. Kemudian *Controller* melakukan pencocokan MAC dan port pada *table* yang sudah ada.



Gambar 6. openflow

(source:<https://media.neliti.com/media/publications/135445-ID-prototipe-infrastruktur-software-defined.pdf>)

Protokol *OpenFlow* terdiri dari 3 fields yaitu *Header Fields*, *Counter* dan *Action*. *Header fields* adalah sebuah *packet header* yang mendefinisikan *flow*, bagian ini bertugas untuk mencocokkan dan membandingkan paket yang masuk ke *OpenVswitch* dengan berdasarkan bidang header. *Counter* adalah sebuah *fields* yang menjaga jumlah paket dan *byte* untuk setiap *flow*, dan melacak berapa jumlah paket yang sesuai dengan *flow*. *Action* adalah *fields* yang mendefinisikan bagaimana paket data akan diproses dan menentukan apa yang harus dilakukan dengan paket yang masuk ke *OpenVswitch* yang sesuai dengan kriteria atau rule. Berikut merupakan contoh *flow table*.

Tabel 5. flowtable

Switch Port	MAC src	MAC dst	Eth Type	LAN ID	IP src	IP dst	IP Pport	TCP sport	TCP dport	Action
Port 1	00-19-D2-8B-5D-FF	00-16-5D-2A	080	lan1	10.0.0.1	10.0.0.2		1726	80	Port 6

Pada contoh *flow table* diatas dapat dilihat bahwa *flow table* telah membentuk sebuah *flow entry* yang akan digunakan untuk mengirim paket dari *source* ke *destination*. Dalam contoh *flow table* tersebut telah didefenisikan dengan jelas port mana saja yang digunakan dan MAC/IP address dari *source/destination*.

3.2.2 *Multipath routing* dan QoS Management

QoS mengacu kepada kemampuan memberikan pelayanan berbeda kepada lalu lintas jaringan dengan kelas-kelas yang berbeda. Tujuan akhir dari QoS adalah memberikan network service yang lebih baik dan terencana dengan *bandwidth* dan *latency* yang terkontrol. QoS didesain untuk membantu *end user* (client) menjadi lebih produktif dengan memastikan bahwa user mendapatkan performansi yang handal dari aplikasi-aplikasi berbasis jaringan. QoS mengacu pada kemampuan jaringan untuk menyediakan layanan yang lebih baik pada trafik jaringan tertentu melalui teknologi yang berbeda-beda. Ada beberapa alasan mengapa memerlukan QoS, yaitu:

1. Untuk memberikan prioritas untuk aplikasi-aplikasi yang kritis pada jaringan.
2. Untuk memaksimalkan penggunaan investasi jaringan yang sudah ada.
3. Untuk meningkatkan performansi untuk aplikasi-aplikasi yang sensitif terhadap delay, seperti Voice dan Video.
4. Untuk merespon terhadap adanya perubahan-perubahan pada aliran traffic di jaringan

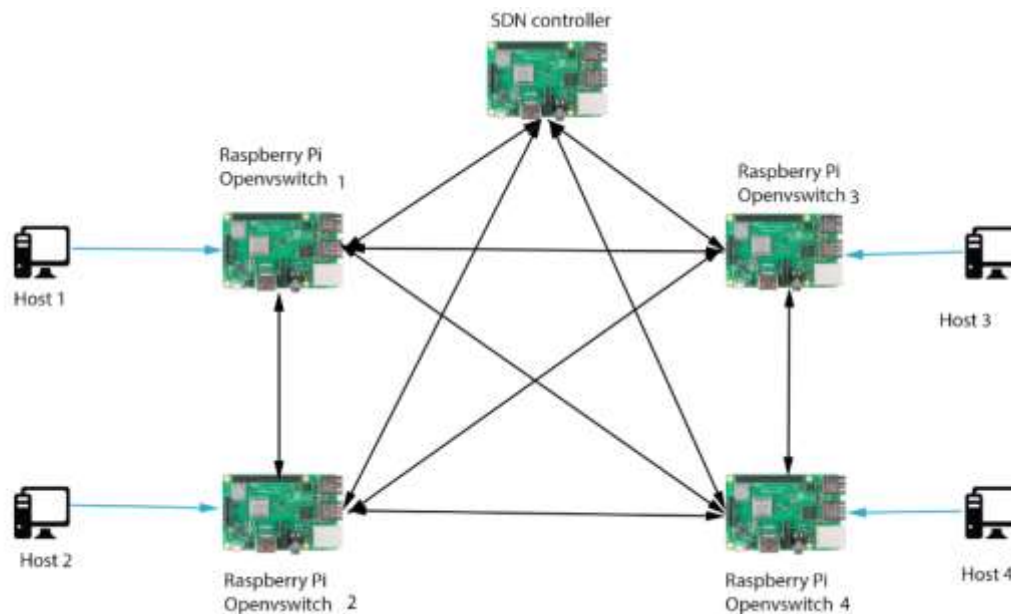
QoS yang umum dipakai adalah *best-effort service*. *Best-effort service* digunakan untuk melakukan semua usaha agar dapat mengirimkan sebuah paket ke suatu tujuan. Penggunaan *best-effort service* tidak akan memberikan jaminan agar paket dapat sampai ke tujuan yang dikehendaki. Sebuah aplikasi dapat mengirimkan data dengan besar yang bebas kapan saja tanpa harus meminta izin atau mengirimkan pemberitahuan ke jaringan.

Didalam QoS management, *Controller* menggunakan *Openflow* untuk membedakan lalu lintas dengan jenis layanan yang berbeda dengan mengidentifikasi alamat IP dari simpul sumber dan untuk memberikan prioritas yang berbeda untuk layanan yang berbeda menggunakan mekanisme antrian pada *OpenVswitch*. Pendekatan ini dapat digunakan untuk berbagai tujuan, termasuk penyeimbangan beban lalu lintas, pengaturan bandwidth untuk meningkatkan throughput, memberikan jaminan end-to-end point dan meningkatkan toleransi kesalahan sistem. *Multipath routing* menggunakan banyak path dalam pengiriman paket dari *source* ke *destination* dan menghitung jalur optimal untuk setiap *flow*. Ketika paket dikirim dan tiba pada *OpenVswitch* yang tidak memiliki flow entry yang cocok, maka akan dikirim pesan ke kontroler untuk memilih jalur optimal dari beberapa jalur yang tersedia di jaringan dan memasukkan flow entri yang baru ke *OpenVswitch* untuk paket forwarding.

Solusi QoS ini memastikan bahwa paket data selalu diteruskan ke jalur yang paling optimal oleh komponen *Multipath routing*. Setelah menemukan jalur optimal, maka perlu dilakukan pengukuran pada *cost path* seperti bandwidth, jumlah hop, delay, dan paket *loss* adalah elemen umum yang dapat digunakan untuk memilih jalur sesuai dengan permintaan *flow*. Dengan mengeksplorasi kinerja QoS di *Multipath routing* dapat mengurangi delay dan paket loss dari *flow* serta dapat meningkatkan pemanfaatan sumber daya *data plane*.

3.2.3 Perancangan Perangkat Keras

Pada bagian ini dijelaskan gambaran rangkaian yang digunakan dalam implementasi *Multipath Routing In SDN Testbed*. Gambaran ini dirancang untuk memberikan gambaran mengenai proses pembuatan rangkaian.



Gambar 7. Rancangan Umum

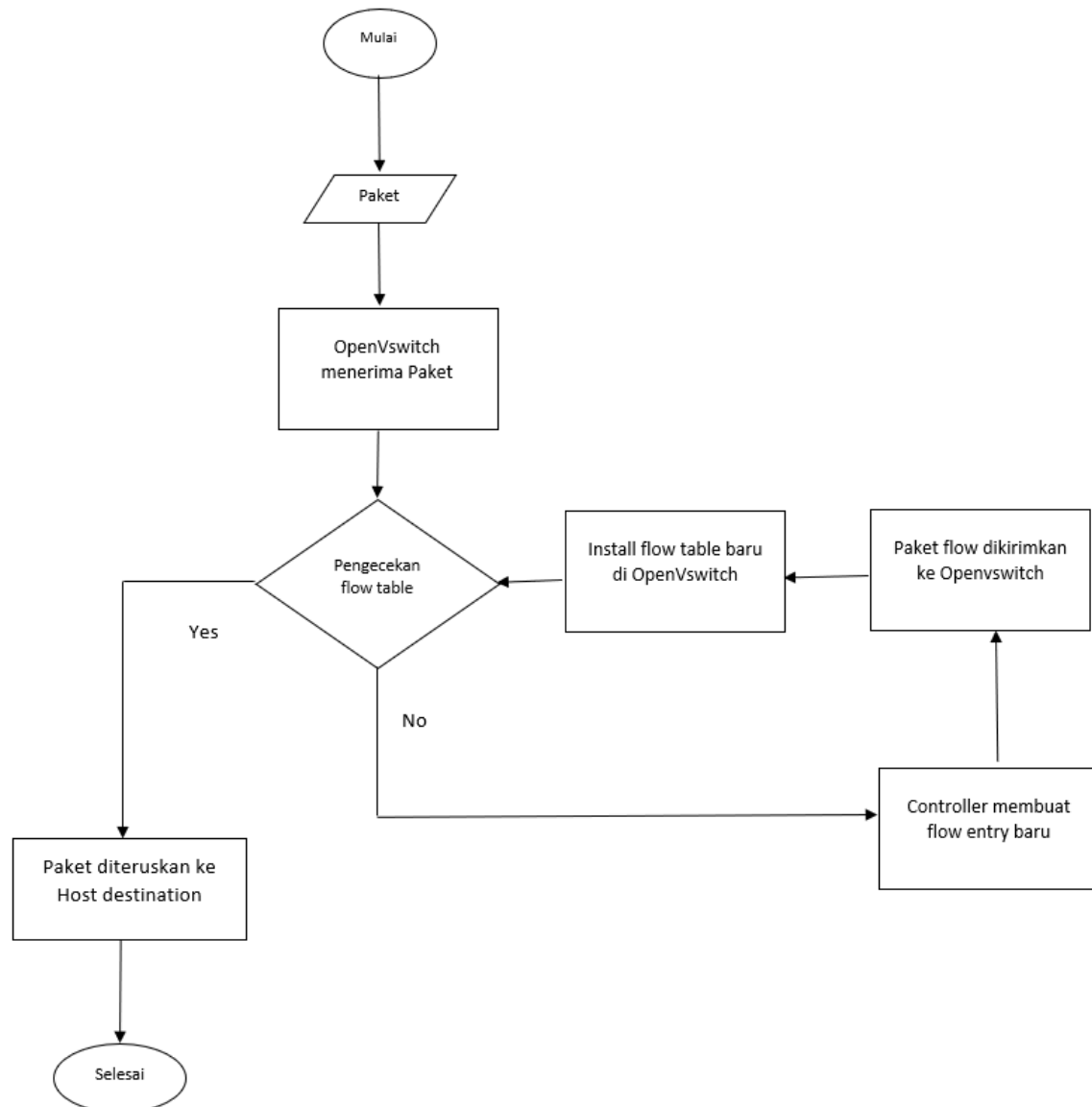
Pada gambar 7 ditampilkan gambaran umum dari sistem *multipath routing in SDN testbed* yang akan dibangun. Raspberry pi berperan sebagai *Controller* dan *OpenVswitch*. Kontroler yang digunakan adalah kontroler ryu yg berbasis python yang akan fokus untuk mengatur aliran (flow) paket secara logik dan menempatkan flow table tersebut pada perangkat *routing* lainnya. *OpenVswitch* akan fokus untuk mengalirkan paket menuju perangkat jaringan lainnya sesuai dengan flow yang diberikan oleh kontroler. Dengan keunggulan SDN ini, *Multipath* pada *Multipath routing* dapat dihitung secara efisien untuk mengoptimalkan penempatan flow table pada perangkat *routing* lainnya. *Host* berfungsi sebagai pengirim dan penerima data ,data yang dikirim nantinya akan ditentukan jalur tercepat oleh kontroler dan akan diteruskan oleh *OpenVswitch* ke penerima . Penghubung antara kontroler dan *OpenVswitch* adalah Kabel LAN , dikarenakan raspberry memiliki satu port untuk LAN/Ethernet maka untuk menghubungkan banyak kabel LAN pada 1 raspberry digunakan USB to Ethernet .

jika *host* yang terhubung dengan *OpenVswitch*, maka *OpenVswitch* mengidentifikasi entri-aliran mana yang cocok. Jika *OpenVswitch* tidak dapat menemukan entri, maka *OpenVswitch* mengirim pesan permintaan ke pengontrol SDN berbasis RYU Jaringan yang dapat diprogram dapat secara dinamis disediakan dengan menyusun rute aliran. Sebagai contoh, paket dapat dirutekan melalui simpul fisik untuk layanan tertentu atau jalur dengan muatan ringan berdasarkan prioritasnya. Cara untuk menyediakan jaringan yang dapat diprogram adalah membuat metrik QoS, atau membuat rute lalu lintas secara manual dengan kontrol pengguna. Ketika persyaratan pengguna yang diminta ini dipenuhi, respons pengontrol Ryu merespons pesan ke *OpenVswitch* ,Ini memperbarui entri aliran dan mentransmisikan ke *host* tujuan.

3.2.4 Flowchart Sistem

1. Flowchart sistem secara umum

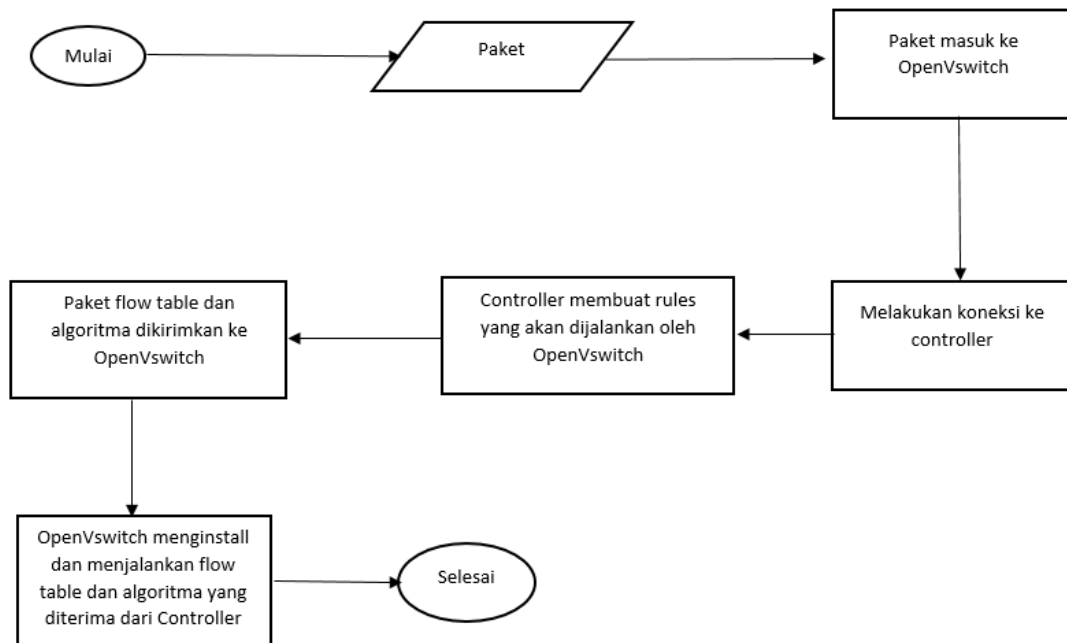
Flowchart ini menggambarkan langkah-langkah proses pengiriman paket data dari *source* ke *destination* mengikuti alur koneksi antara *OpenVswitch* dan *Controller*



Gambar 8. flowchart secara umum

2. Flowchart *OpenVswitch*

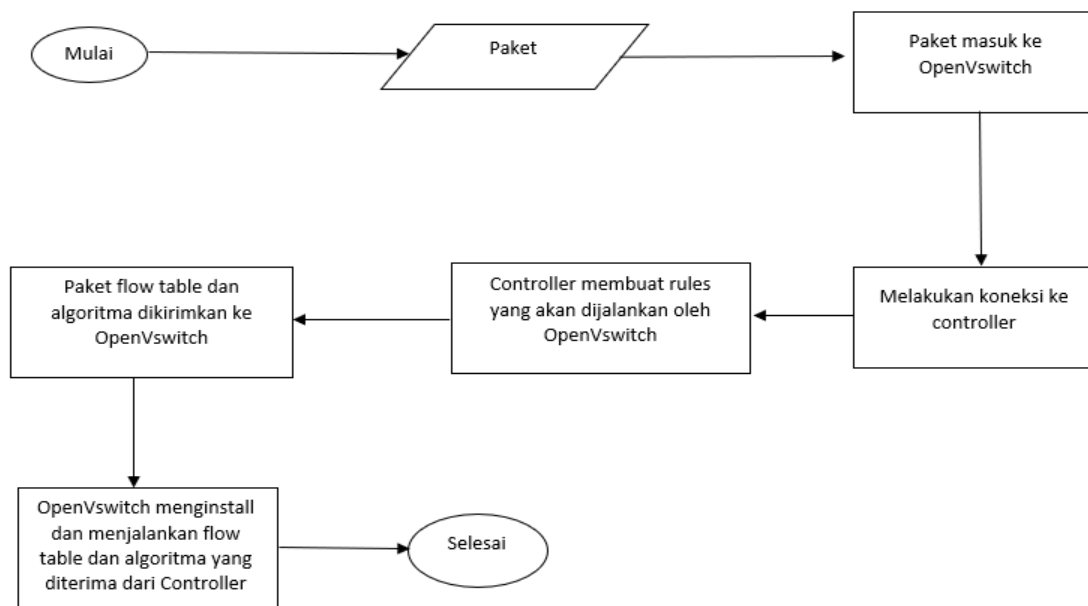
Flowchart ini menunjukkan langkah-langkah proses kerja *OpenVswitch* sehingga dapat menjalankan rules dari *Controller*



Gambar 9. flowchart *OpenVswitch*

3. Flowchart *Controller*

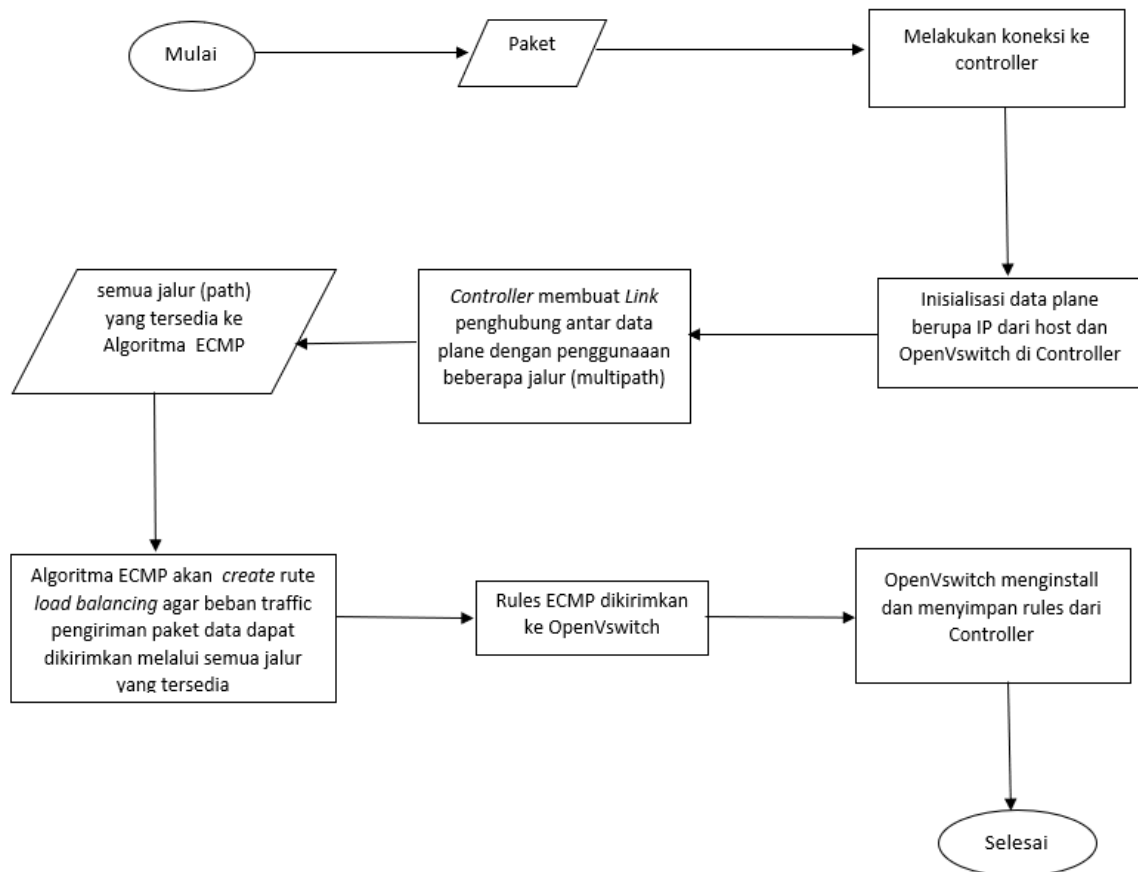
Flowchart ini menggambarkan proses pembuatan rules di *Controller* yang nantinya akan dijalankan oleh *OpenVswitch*.



Gambar 10. flowchar *Controller*

4. Flowchart Algoritma ECMP

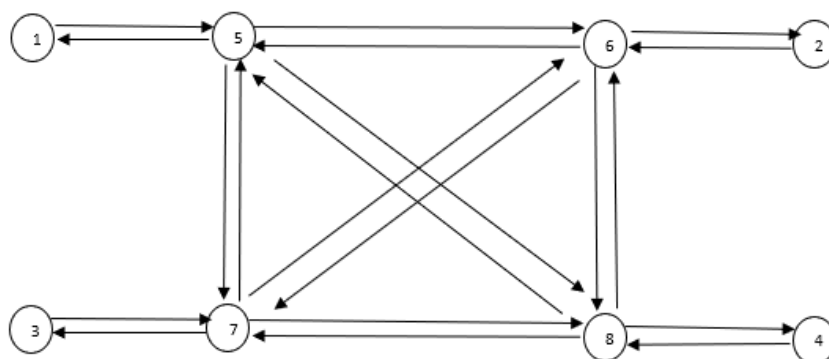
Flowchart ini menunjukkan langkah-langkah algoritma ECMP berjalan sehingga proses pembentukan algoritma ECMP dalam mengimplementasikan *load balancing* dengan *Multipath routing*.



Gambar 11. flowchart Algoritma ECMP

3.2.4 Graf Algoritma ECMP

Pada graf ini akan ditunjukkan jalur mana saja yang dapat digunakan setika ada pengiriman data dari *source* ke *destination* menggunakan *Multipath routing*



Gambar 12. graf ECMP

Keterangan:

- 1 adalah *host 1*
- 2 adalah *host 2*
- 3 adalah *host 3*
- 4 adalah *host 4*
- 5 adalah *OpenVswitch 1*
- 6 adalah *OpenVswitch 2*
- 7 adalah *OpenVswitch 3*
- 8 adalah *OpenVswitch 4*

Dari graf diatas misalkan Host 1 akan mengirimkan paket data ke host 4, maka akan menggunakan *multipath* routing jadi kemungkinan jalur yang dapat dilalui untuk mengirimkan paket data ada lebih dari satu jalur. Pada skenario ini, setiap path akan mendapat besar bandwidth yang sama. Kemudian fungsi dari algoritma ECMP adalah untuk menyeimbangkan setiap jalur yang tersedia sehingga pengiriman paket data dapat dilakukan secara maksimal. Disini algoritma ECMP dapat menggunakan 5 jalur yang tersedia untuk mengirimkan paket data dari *source* ke *destination* yaitu 1->5->8->4; 1->5->6->8->4; 1->5->7->8->4; 1->5->7->6->8->4; dan 1->5->6->7->8->4.

3.3 Skenario Pengujian

Pada pengerjaan tugas akhir ini akan dilakukan beberapa simulasi pengujian yang nantinya akan diimplementasikan pada *Multipath routing* in SDN testbed. Pada skenario uji ini terdapat beberapa komponen penting seperti Raspberry pi sebagai *OpenVswitch* dan *Controller*, kabel LAN Sebagai penghubung jaringan antara *Controller*, penghubung antara *OpenVswitch* dan *host*, serta USB to LAN Adapter untuk menghubungkan Lan/ Ethernet melalui port USB. Nantinya akan dilakukan pengukuran berupa throughput, dan delay ketika dilakukan pengiriman data dari *source* ke *loss* menggunakan alat ukur yang didasarkan pada fungsi *tools* Iperf dan Packet Internet Groper (ping). Iperf adalah alat pengujian kinerja jaringan yang dapat membuat aliran UDP atau TCP antara *host* klien dan *server host*, *bandwidth real-time(throughput)*, *packet loss*, dan *delay* dapat ditampilkan ketika Iperf beroperasi. Ping dapat menggunakan (*Internet Control Messages Protocol*) ICMP untuk menghitung *round-trip time* paket yang menunjukkan *respon time* dan informasi *delay*.

3.3.1 Pengujian *Multipath Routing* dalam No-limited Bandwidth Topologi

Salah satu tujuan dari pengerjaan TA ini adalah untuk meningkatkan kinerja jaringan dengan mengimplementasikan *Multipath routing*. Penggunaan *Multipath routing* berdasarkan SDN untuk meningkatkan *load balancing* dan untuk meningkatkan *throughput* sambil mengurangi kemacetan dalam pengiriman data secepat mungkin. Tugas Akhir ini juga menggunakan pengukuran menggunakan metode *shortest single-path routing* sebagai perbandingan dengan metode pengukuran menggunakan metode *Multipath routing*. Throughput sebagai indikator penting dari kinerja jaringan adalah ukuran total data yang berhasil ditransmisikan dalam interval satuan waktu. Dalam pengujian ini digunakan nilai pengiriman trafik data dengan kecepatan data yang sama dan waktu yang sama. Pengujian ini juga tidak menentukan nilai maksimum bandwidth yang dapat digunakan oleh setiap *host* pada setiap *path* yang menjadi jalur pengiriman data antar *host*. Karena *singlepath routing* hanya meneruskan paket di jalur terpendek antara sumber dan tujuan, maka 2 *host* pada setiap *OpenVswitch* akan berbagi bandwidth jalur yang sama sehingga dapat

menyebabkan kemacetan dan *delay* yang besar ketika terjadi pengiriman paket data. Hasil dari penggunaan metode *singlepath* dan *Multipath routing* inilah yang nantinya akan dibandingkan dengan mengukur nilai *throughput*, *paket loss*, dan *delay* [22].

3.3.2 Pengujian QoS dalam Limited Bandwidth Topologi

QoS menjadi salah satu faktor penguji untuk membuktikan bahwa dengan penerapan *Multipath routing* dapat meningkatkan kinerja dari jaringan. Oleh karena itu dilakukan pengukuran QoS pada level *best-effort service* untuk mendapatkan hasil yang diinginkan. Pengujian ini masih menggunakan topologi yang sama dalam Gambar.6 untuk melakukan pengujian. Pengujian ini juga menetapkan nilai maksimum bandwidth yang dapat digunakan oleh setiap *host* pada setiap *path* yang menjadi jalur pengiriman data antar *host*. Misalnya terdapat 2 *host* dalam setiap *OpenVswitch* dengan nilai maksimum bandwidth yang dapat digunakan adalah 2Gbit/s. Karena kontroler yang digunakan memberikan batas minimum pada penggunaan bandwidth sebesar 2Gbit/s. Jika membatasi besar bandwidth sebesar 1Gbit/s atau 100Mbit/s akan berdampak pada jumlah besar *throughput* pada *singlepath* dan *multipath* sehingga akan memberikan hasil yang sama karena tidak mendapatkan perbedaan yang signifikan, sehingga jika diberikan nilai *bandwidth* yang besar akan membuat hasil *throughput* antara *singlepath* dan *multipath* akan berbeda. Dan pada pengujian kedua tiap paket diberi nilai yang sama dan berbeda-beda, seperti pada skenario pertama paket dan bandwidth diberi nilai yang sama, paket tiap host diberi nilai sebesar 500Mbit/s dan besar bandwidth sebesar 2Gbit/s. Pada skenario kedua besar bandwidth tiap jalur sama dan besar paket tiap host berbeda-beda. Namun berbeda dengan pengujian sebelumnya, total nilai bandwidth pada keseluruhan path tidak diatur (*default*). Hal ini dapat menyebabkan pengiriman paket bisa terkendala atau terjadi *congestion*, namun hal ini dapat diatasi dengan penggunaan beberapa jalur pengiriman data atau penggunaan metode *load balancing* sehingga dapat mengurangi *congestion*. Meskipun kemacetan dapat dikendalikan dengan mengalokasikan lalu lintas ke jalur yang berbeda dengan penggunaan sumber daya jaringan fisik yang lebih baik, *gateway* tetap saja membatasi total *throughput* sistem pada setiap *host*. Oleh karena itu dilakukan pengukuran QoS pada level *best-effort* untuk mengukur nilai *throughput*, *paket loss*, dan *delay* sehingga akan didapatkan hasil sesuai yang diharapkan [22].

BAB IV IMPLEMENTASI DAN PENGUJIAN

Pada bab ini dijelaskan mengenai implementasi dan pengujian terhadap sistem yang telah dilakukan. Proses implementasi dan pengujian yang dilakukan berupa instalasi, konfigurasi, implementasi prototipe, dan pengujian prototipe disimulator dan perancangan pada testbed.

4.1 Implementasi dan Pengujian *Singlepath* dengan *Multipath routing* di Simulator

Pada subbab ini diuraikan tentang proses instalasi yang dilakukan dalam pengerjaan implementasi *Singelpath* dan *Multipath routing di SDN* simulator. Instalasi yang dilakukan seperti instalasi simulator mininet dan ryu sebagai controller pada jaringan SDN yang dibangun. Berikut akan dijelaskan mengenai instalasi yang akan dilakukan.

4.1.1 instalasi mininet

Mininet adalah sebuah *software open source* yang digunakan untuk pengujian dan simulasi jaringan mengenai SDN. Mininet dapat menggunakan kernel dari linux sehingga dapat menjalankan jaringan SDN yang sebenarnya dan lebih ringan dan cepat. Berikut adalah proses instalasi mininet di sistem operasi linux.

1. Instalasi git

```
$ sudo apt-get install git
```

2. Selanjutnya clone repository mininet dari github

```
$ git clone git://github.com/mininet/mininet
```

3. Lakukan instalasi menggunakan .sh agar seluruh package yang dibutuhkan mininet dapat terinstal dan menggunakan opsi -a agar mininet dapat menginstal mininet secara kompleks seperti Openvswitch sudah termaksud didalamnya.

```
$ mininet/util/install.sh -a
```

4. Setelah instalasi selesai maka coba menjalankan perintah dibawah ,jika semua terhubung maka mininet telah terinstal.

```
$ sudo mn
```

4.1.2 Instalasi Ryu di simulator

Ryu merupakan controller yang didefinisikan menggunakan bahasa pemrograman python sehingga sangat memudahkan untuk mengontrol jaringan ,pada SDN ryu digunakan sebagai controller yang akan mengatur openflow pada jaringan SDN. Berikut instalasi ryu di simulator.

1. Install python menggunakan pip, karena ryu merupakan platform berbasis python

```
$ sudo apt-get install python-pip
```

2. Install requirement software untuk Ryu

```
$ sudo apt-get install gcc python-dev libffi-dev libssl-dev libxml2-dev libxslt1-dev zlib-dev
```

3. Instal pip

```
$ sudo apt-get install pip
```

4. Install ryu menggunakan pip

```
$ pip install ryu
```

5. Setelah selesai dan tidak terjadi error maka jalan kan perintah berikut jika sudah muncul pada CLI maka instalasi telah selesai

```
$ ryu-manager
```

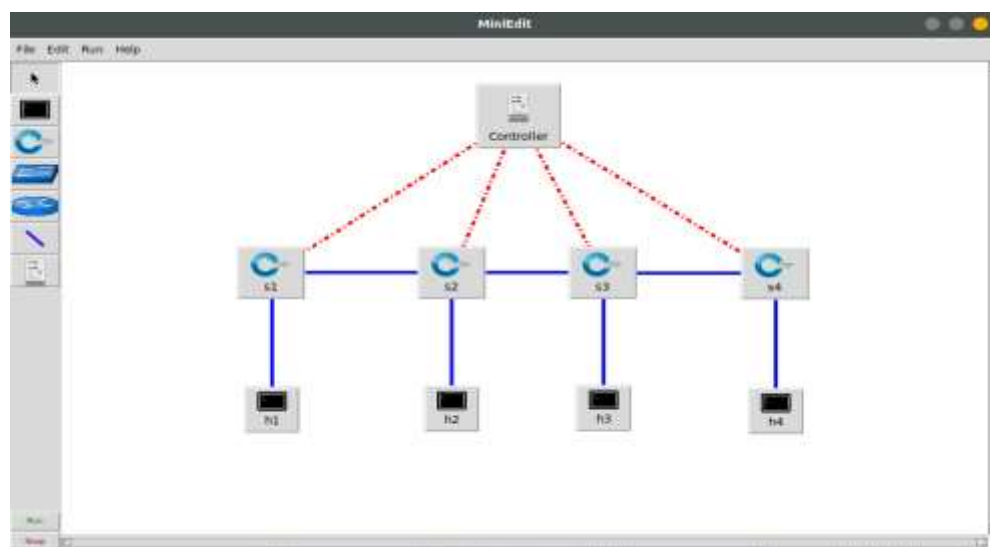
4.2 Pengujian SDN di Mininet

Pada sub bab ini diuraikan mengenai implementasi yang dilakukan dalam eksperimen. Implementasi *Multipath routing* SDN di dalam mininet, sebelum masuk dalam implementasi *Multipath routing* di SDN testbed di *raspberrypi* penulis menjelaskan terlebih dahulu didalam simulator yaitu mininet. Implementasi *singlepath* dan *Multipath routing* di dalam mininet dapat dilihat pada penjelasan di bawah ini.

4.2.1 Singlepath Routing SDN di Mininet

Pada sub bab ini diuraikan implementasi *singlepath routing* di mininet. Implementasi *singlepath* dan *Multipath routing* di dalam mininet dapat dilihat pada gambar di bawah ini.

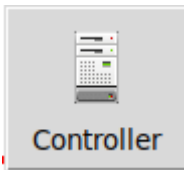


a. Topologi *Singlepath* di miniedit



Gambar 13. Topologi *Singlepath routing*

Pada gambar diatas menjelaskan bahwa topologi memiliki 1 kontroler 4 *OpenVswitch* dan 4 *host* pada miniedit di mininet. Setiap pengiriman dari *host* 1 ke *host* 4 akan melalui 1 path saja tidak dapat memilih path yang lain sehingga jika terjadi traffic yang tinggi maka bandwidth atau lebar jalur pengiriman akan semakin mengecil dan akan terjadi drop pada paket sehingga tidak dapat sampai pada tujuan.

Tabel 6 Keterangan Topologi Miniedit

No	Gambar	Berfungsi	Keterangan
1.		Kontroler	Sebagai kontroler pada topology yang akan mengatur dan memberikan <i>openflow</i> pada jaringan
2.		<i>OpenVswitch</i>	Sebagai <i>OpenVswitch</i> yang menjalankan <i>openflow</i> yang diberikan kontroler
3		<i>Host</i>	Sebagai <i>host</i> yang dapat mengirim paket ke satu atau banyak paket

b.Topologi dalam pemrograman python

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.link import TCLink,Link
```

```
class Topologi(Topo):
```

```
    def __init__(self):
```

```
        Topo.__init__(self)
```

```
        h1 = self.addHost('h1', mac='00:00:00:00:01:01',ip='10.0.0.1/24')
```

```
        h2 = self.addHost('h2', mac='00:00:00:00:01:02',ip='10.0.0.2/24')
```

```
        h3 = self.addHost('h3', mac='00:00:00:00:01:03',ip='10.0.0.3/24')
```

```
        h4 = self.addHost('h4', mac='00:00:00:00:01:04',ip='10.0.0.4/24')
```

```
        s1 = self.addSwitch('s1',mac='00:00:00:00:00:01')
```

```
        s2 = self.addSwitch('s2',mac='00:00:00:00:00:02')
```

```
        s3 = self.addSwitch('s3',mac='00:00:00:00:00:03')
```

```
        s4 = self.addSwitch('s4',mac='00:00:00:00:00:04')
```

```

self.addLink(s1,h1)
self.addLink(s2,h2)
self.addLink(s3,h3)
self.addLink(s4,h4)

self.addLink(s1,s2,cls=TCLink,
delay='1ms',loss=0,max_queue_size=1000,use_htb=True)
self.addLink(s2,s3,cls=TCLink,
delay='1ms',loss=0,max_queue_size=1000,use_htb=True)
self.addLink(s3,s4,cls=TCLink,
delay='1ms',loss=0,max_queue_size=1000,use_htb=True)

topos = {'switch4':(lambda:Topologi())}

```

Pada kode diatas merupakan program untuk membentuk topologi *singlepath* sama dengan bentuk pada gambar 12 diatas. dalam mininet bebas membentuk topologi dari mana saja bisa melalui miniedit bisa juga dibentuk dalam pemrograman python . Pada Tugas akhir ini penulis memilih membentuknya dalam pemrograman python karena lebih bebas dalam mengatur topologi dan sangat fleksibel dan codingnya mudah untuk dimengerti .

c. Kontroler pada *singlepath*

Dalam topologi diperlukan kontroler untuk dapat memberikan flow pada setiap path akan semakin topologi hanya akan membangun arsitektur jaringan namun tidak memberikan flow atau pun tidak akan membuat setiap *node* menjadi terhubung maka disini kontroler sebagai otak dari jaringan dan otot nya adalah switch nya . Dalam kode dibawah ini dapat dilihat potongan kodingan kontroler dari *singlepath*.

```

class SimpleSwitch(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_0.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch, self).__init__(*args, **kwargs)
        self.mac_to_port = { }

    def add_flow(self, datapath, in_port, dst, src, actions):
        ofproto = datapath.ofproto

        match = datapath.ofproto_parser.OFPMatch(
            in_port=in_port,
            dl_dst=haddr_to_bin(dst), dl_src=haddr_to_bin(src))

        mod = datapath.ofproto_parser.OFPFlowMod(
            datapath=datapath, match=match, cookie=0,
            command=ofproto.OFPFC_ADD, idle_timeout=0, hard_timeout=0,
            priority=ofproto.OFP_DEFAULT_PRIORITY,
            flags=ofproto.OFPFF_SEND_FLOW_REM, actions=actions)
        datapath.send_msg(mod)

```

```

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto

    pkt = packet.Packet(msg.data)
    eth = pkt.get_protocol(ethernet.ethernet)

    if eth.ethertype == ether_types.ETH_TYPE_LLDP:
        # ignore lldp packet
        return
    dst = eth.dst
    src = eth.src

    dpid = datapath.id
    self.mac_to_port.setdefault(dpid, {})

    self.logger.info("packet in %s %s %s %s", dpid, src, dst, msg.in_port)

    # learn a mac address to avoid FLOOD next time.
    self.mac_to_port[dpid][src] = msg.in_port

    if dst in self.mac_to_port[dpid]:
        out_port = self.mac_to_port[dpid][dst]
    else:
        out_port = ofproto.OFPP_FLOOD

    actions = [datapath.ofproto_parser.OFPActionOutput(out_port)]

    # install a flow to avoid packet_in next time
    if out_port != ofproto.OFPP_FLOOD:
        self.add_flow(datapath, msg.in_port, dst, src, actions)

    data = None
    if msg.buffer_id == ofproto.OFP_NO_BUFFER:
        data = msg.data

    out = datapath.ofproto_parser.OFPPacketOut(
        datapath=datapath, buffer_id=msg.buffer_id, in_port=msg.in_port,
        actions=actions, data=data)
    datapath.send_msg(out)

```

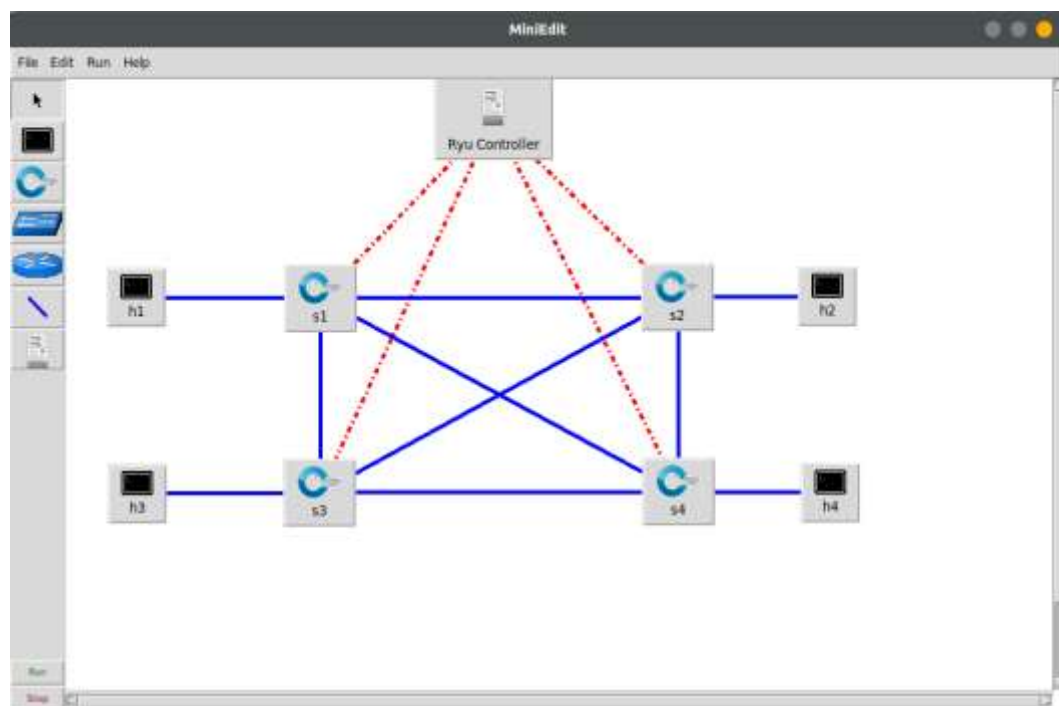
Pada setiap skenario menunjukkan bahwa terjadinya penurunan performa jaringan dan lebar bandwidth apabila path yang dilalui dipakai oleh setiap *host* dan pengiriman paket dengan cara bersamaan untuk meningkatkan trafik dari tiap jalur yg dilalui.

Dari bagan tabel dapat dilihat bahwa terjadi penurunan performa jaringan pada setiap path yang dilalui karena hanya 1 jalur yang dapat dilalui oleh paket sehingga pada saat *host* yang

lain mengirimkan paket melalui jalur yang sama maka akan terjadi trafik yang tinggi pada jalur tersebut dan membuat lebar jalur tersebut akan menurun. Dari tabel diatas dapat dilihat bahwa tidak terjadi peningkatan performa dari *singlepath* .

4.2.2 *Multipath Routing* SDN di Mininet

Pada sub bab ini diuraikan mengenai implementasi yang dilakukan dalam eksperimen. Implementasi *Multipath routing* SDN di dalam mininet, sebelum masuk dalam implementasi *Multipath routing* di SDN testbet di *raspberry pi*, penulis akan menjelaskan terlebih dahulu didalam simulator yaitu mininet .Dalam implementasi *Multipath* di mininet penulis akan menguji nya dengan beberapa algoritma dalam pengiriman paket antar *host* dan algoritma untuk menghitung cost antar path itu sendiri didalam sebuah kontroler itu sendiri. Implementasi *Multipath routing* di dalam mininet dapat dilihat pada penjelasan di bawah ini.



Gambar 14. Topologi *Multipath routing*

Pada gambar diatas menjelaskan bahwa topologi memiliki 1 kontroler 4 *OpenVswitch* dan 4 *host* pada miniedit di mininet. Setiap *OpenVswitch* sudah saling terhubung satu sama lain tanpa harus terhubung ke switch lain agar dapat melakukan pengiriman paket antar *host* . Dan lebih banyak memilih path yang akan dilalui nya path dan dapat memperhitungkan besar cost setiap jalur yang akan dipilih oleh *host* jika ingin mengirimkan paket ke *host* lain nya.

a. Topologi *Multipath routing* dalam pemrograman python menjalankan topologi di mininet dengan cara berikut :

```
root@desktop-13-1311:/home/TA/mininet/custom# sudo mn --custom [file_topologi].py --topo switch4 --mac --switch ovsk --Controller=remote
```

```

class Topologi(Topo):

    def __init__(self):

        Topo.__init__(self)

        h1 = self.addHost('h1', mac='00:00:00:00:01:01',ip='10.0.0.1/24')
        h2 = self.addHost('h2', mac='00:00:00:00:01:02',ip='10.0.0.2/24')
        h3 = self.addHost('h3', mac='00:00:00:00:01:03',ip='10.0.0.3/24')
        h4 = self.addHost('h4', mac='00:00:00:00:01:04',ip='10.0.0.4/24')

        s1 = self.addSwitch('s1',mac='00:00:00:00:00:01')
        s2 = self.addSwitch('s2',mac='00:00:00:00:00:02')
        s3 = self.addSwitch('s3',mac='00:00:00:00:00:03')
        s4 = self.addSwitch('s4',mac='00:00:00:00:00:04')

        self.addLink(s1,h1)
        self.addLink(s2,h2)
        self.addLink(s3,h3)
        self.addLink(s4,h4)

        self.addLink(s1,s2,cls=TCLink,bw=7000,
delay='1ms',loss=0,max_queue_size=1000,use_htb=True)
        self.addLink(s2,s3,cls=TCLink,bw=7000,
delay='1ms',loss=0,max_queue_size=1000,use_htb=True)
        self.addLink(s3,s4,cls=TCLink,
bw=7000,delay='1ms',loss=0,max_queue_size=1000,use_htb=True)
        self.addLink(s4,s1,cls=TCLink,bw=7000,
delay='1ms',loss=0,max_queue_size=1000,use_htb=True)
        self.addLink(s1,s3,cls=TCLink,bw=7000,
delay='1ms',loss=0,max_queue_size=1000,use_htb=True)
        self.addLink(s2,s4,cls=TCLink,bw=7000,
delay='1ms',loss=0,max_queue_size=1000,use_htb=True)

        topos = {'switch4':(lambda:Topologi())}

```

Pada gambar diatas merupakan program untuk membentuk topologi *Multipath* sama dengan bentuk pada diatas. Pada gambar diatas dapat dibentuk suatu topologi jaringan dan bebas untuk menghubungkan antar *node* pada setiap switch sehingga membentuk topologi yang diinginkan . Penulis juga mengatur besar bandwidth delay dan ip address dari topologi , namun bukan hanya dalam topologi dapat mengatur bandwidth ,delay, dan ip address namun pada kontroler juga dapat mengaturnya .

b. Kontroler pada *Multipath*

Dalam topologi diperlukan kontroler yang merupakan komponen yang paling penting dalam sebuah jaringan SDN , untuk membangun jaringan SDN maka kontroler merupakan otak dari jaringan yang akan mengatur semua flow pada jaringan. Dalam kontroler dimuat algoritma untuk flow dari jaringan . Dibawah ini dapat dilihat potongan kodingan kontroler dari *Multipath* cara menjalankan kontroler pada mininet.

```
root@desktop-13-1311:/home/TA# ryu-manager ryu/ryu/app/ryu_Multipath.py --observe-links
```

```
class ProjectController(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(ProjectController, self).__init__(*args, **kwargs)
        self.mac_to_port = { }
        self.topology_api_app = self
        self.datapath_list = { }
        self.arp_table = { }
        self.switches = []
        self.hosts = { }
        self.Multipath_group_ids = { }
        self.group_ids = []
        self.adjacency = defaultdict(dict)
        self.bandwidths = defaultdict(lambda: defaultdict(lambda: DEFAULT_BW))

    def get_paths(self, src, dst):
        """
        Get all paths from src to dst using DFS algorithm
        """
        if src == dst:
            # host target is on the same switch
            return [[src]]
        paths = []
        stack = [(src, [src])]
        while stack:
            (node, path) = stack.pop()
            for next in set(self.adjacency[node].keys()) - set(path):
                if next is dst:
                    paths.append(path + [next])
                else:
                    stack.append((next, path + [next]))
        print "Available paths from ", src, " to ", dst, " : ", paths
        return paths

    def get_link_cost(self, s1, s2):
        """
        Get the link cost between two switches
        """
        e1 = self.adjacency[s1][s2]
        e2 = self.adjacency[s2][s1]
        b1 = min(self.bandwidths[s1][e1], self.bandwidths[s2][e2])
        ew = REFERENCE_BW/b1
        return ew
```

```

def get_path_cost(self, path):
    """
    Get the path cost
    """
    cost = 0
    for i in range(len(path) - 1):
        cost += self.get_link_cost(path[i], path[i+1])
    return cost

```

Dalam kontroler terdiri dari 3 jenis algoritma untuk membangun flow dalam jaringan adalah sebagai berikut:

1. ECMP

ECMP memungkinkan penggunaan beberapa jalur yang memiliki nilai cost yang sama dalam perutean. Fitur ini bukan hanya akan membantu mendistribusikan lalu lintas lebih merata, tetapi juga sebagai metode proteksi atau keamanan dalam jaringan. Dengan menggunakan ECMP, jalur yang memiliki nilai yang sama akan dipasang ke dalam table load balancing di lapisan forwarding pada router. Potongan kodingan dapat dilihat pada *ryu_Multipath.py*

```

@set_ev_cls(event.EventSwitchEnter)
def switch_enter_handler(self, ev):
    switch = ev.switch.dp
    ofp_parser = switch.ofproto_parser

    if switch.id not in self.switches:
        self.switches.append(switch.id)
        self.datapath_list[switch.id] = switch

    # Request port/link descriptions, useful for obtaining bandwidth
    req = ofp_parser.OFPPortDescStatsRequest(switch)
    switch.send_msg(req)

@set_ev_cls(event.EventSwitchLeave, MAIN_DISPATCHER)
def switch_leave_handler(self, ev):
    print ev
    switch = ev.switch.dp.id
    if switch in self.switches:
        self.switches.remove(switch)
        del self.datapath_list[switch]
        del self.adjacency[switch]

@set_ev_cls(event.EventLinkAdd, MAIN_DISPATCHER)
def link_add_handler(self, ev):
    s1 = ev.link.src
    s2 = ev.link.dst
    self.adjacency[s1.dpid][s2.dpid] = s1.port_no
    self.adjacency[s2.dpid][s1.dpid] = s2.port_no

```

4.3 Pengujian Antara *Singlepath* dengan *Multipath routing*

Untuk melihat perbandingan performansi jaringan antara *singlepath* dan *multipath* routing maka penulis melakukan Pengujian Antara *Singlepath* dan *Multipath* dilakukan dengan menggunakan 4 skenario. Untuk melihat skenario dapat dilihat pada penjelasan berikut.

1. Skenario 1

pada skenario ini dilakukan pengujian dengan menggunakan *singlepath* dan *multipath* dimana besar paket yang dikirimkan antar setiap host adalah sama yaitu 500MB dan lebar bandwidthnya juga sama yaitu sebesar 2 Gbits. Hasil yang didapatkan setelah dilakukan pengujian pada skenario 1, dapat dilihat pada tabel 7.

Tabel 7. Skenario 1

No	Host	Besar Paket	Band width	Throughput (Singlepath) Mbits	Throughput (Multipath) Mbits	Delay (Singlepath)	Delay (multipath)	Paket Loss (singlepath)	Paket Loss(multipath)
1	Host2->Host1	500MB	2GB	0.820	1.01	0.00	0.07	0	0
2	Host3->Host1	500MB	2GB	0.813	1.058	0.00	0.14	0	0
3	Host4->Host1	500MB	2 GB	0.671	1.3	0.00	0.14	0	0
4	host1->host2	500MB	2GB	0.662	1.98	0.00	0.19	0.1	0
5	host3->host2	500MB	2GB	0.875	1.87	0.00	0.04	0	0
6	host4->host2	500MB	2 GB	0.917	1.99	0.18	0.05	0.4	0
7	host1->host3	500MB	2GB	0.872	1.93	0.01	0.19	0.2	0
8	host2->host3	500MB	2GB	0.872	1.99	0.00	0.04	1	0
9	host4->host3	500MB	2 GB	0.762	1.91	0.00	0.03	0.2	0
10	host1->Host4	500MB	2GB	0.890	1.85	0.00	0.03	1	0
11	host2->Host4	500MB	2GB	0.865	1.83	0.00	0.15	1	0
12	host3->Host4	500MB	2 GB	0.880	1.76	0.001	0.00	0.01	0
TOTAL				0,825	1.70	0.01	0.09	0.32	0

2. Skenario 1.1

pada skenario ini penulis menguji antara *singlepath* dan *multipath* dengan besar paket yang dikirimkan antar setiap host tersebut berbeda dan lebar bandwidth nya sama yaitu sebesar 2Gbits. Untuk Hasil yang didapatkan setelah dilakukan pengujian pada skenario 2, dapat dilihat pada table 8.

Tabel 8. Skenario 1.1

No	Host	Besar Paket	Band width	Throughput(<i>Singlepath</i>)G bits	Throughput (<i>Multipath</i>)Gbit	Delay (<i>singelpath</i>)	Delay (<i>Multipath</i>)	Paket Loss (<i>singlepat</i>)	Paket Loss(<i>mul tipath</i>)
1	Host2->Host1	100M B	2GB	0,819	1.09	0	0	0.2	0
2	Host3->Host1	750M B	2GB	0,894	1.29	0.43	0	1	0
3	Host4->Host1	1GB	2 GB	0,908	1.34	0.13	0.01	0	0
4	host1->host2	100M B	2GB	0,819	1.48	0.18	0.05	0.1	0
5	host3->host2	750M B	2GB	0,894	1.43	0.45	0	0	0
6	host4->host2	1GB	2 GB	0,908	1.81	0.02	0	0.4	0
7	host1->host3	100M B	2GB	0,796	1.67	0	0.01	0.2	0
8	host2->host3	750M B	2GB	0,889	1.12	0.02	0.06	1	0
9	host4->host3	1GB	2 GB	0,906	1.81	0.06	0.05	0.2	0
10	host1->Host4	100M B	2GB	0,852	1.58	0.06	0	1	0
11	host2->Host4	750M B	2GB	0,917	1.39	0.17	0	1	0
12	host3->Host4	1GB	2 GB	0,910	1.71	0.03	0	0.01	0
TOTAL				0,876	1.47	0.12	0.01	0.42	0

3. Skenario 2

pada skenario ini penulis menguji antara *singlepath* dan *multipath* dengan besar paket yang dikirimkan antar setiap host tersebut adalah sama yaitu 30GB dan lebar bandwidth nya adalah sesuai default sehingga controller yang akan menentukan secara otomatis lebar bandwidth yang akan dipakai untuk mengirimkan paket. Hasil yang didapatkan setelah dilakukan pengujian pada skenario 2, dapat dilihat pada table 9.

Tabel 9. Skenario 2

No	Host	Besar Paket	Bandwidth	Throughput (<i>Singlepath</i>) GBits	Throughput (<i>Multipath</i>) GBits	Delay (<i>singelpath</i>)	Delay(<i>Multi path</i>)	Paket Loss(<i>singlepath</i>)	Paket Loss(<i>multi path</i>)
1	Host2->Host1	30GB	No-limited	17.4	25.9	0.21	0.1	0	0
2	Host3->Host1	30GB	No-limited	15	22	0	0	0	0
3	Host4->Host1	30GB	No-limited	10.4	25.4	0.1	0	0	0

No	Host	Besar Paket	Bandwidth	Throughput (Singlepath) GBits	Throughput (Multipath) GBits	Delay (single path)	Delay(Multi path)	Paket Loss(singlepath)	Paket Loss(multi path)
4	host1->host2	30GB	No-limited	17.1	24.7	0.01	0	0.1	0
5	host3->host2	30GB	No-limited	16.7	22.3	0.4	0	0	0
6	host4->host2	30GB	No-limited	16.8	20.8	0.4	0.04	0	0
7	host1->host3	30GB	No-limited	15.4	20.7	0.2	0.09	0	0
8	host2->host3	30GB	No-limited	8.38	20.8	0.21	0.16	0.1	0
9	host4->host3	30GB	No-limited	16.7	21.1	0.15	0.15	0	0
10	host1->Host4	30GB	No-limited	14.3	20.3	0.12	0	0	0
11	host2->Host4	30GB	No-limited	15.9	20	0.021	0	1	0
12	host3->Host4	30GB	No-limited	17.9	20.9	0.26	0.02	0.01	0
TOTAL				15.1	22	0.17	0.04	0.10	0

4. skenario 2.2

pada skenario ini penulis menguji antara *singlepath* dan *multipath* dengan besar paket yang dikirimkan antar setiap host tersebut berbeda dan lebar bandwidth nya adalah sesuai *default* sehingga *controller* yang akan menentukan secara otomatis lebar *bandwidth* yang akan dipakai untuk mengirimkan paket. Hasil yang didapatkan setelah dilakukan pengujian pada skenario 2.2, dapat dilihat pada tabel 10.

Tabel 10. Skenario 2.2

No	Host	Besar Paket	Bandwith	Throughput (Singlepath)/ GBits	Throughput (Multipath)/ GBits	Delay(single path)	Delay (Multi path)	Paket Loss(single path)	Paket Loss(multi path)
1	Host2->Host1	10GB	No-limited	18.8	22.2	0.4	0	0	0
2	Host3->Host1	15GB	No-limited	16.2	21.6	0.03	0	0	0
3	Host4->Host1	20GB	No-limited	10.6	21.2	0.21	0.1	0	0
4	host1->host2	10GB	No-limited	19.7	20.3	0.19	0	0	0
5	host3->host2	15GB	No-limited	18.4	20.4	0.19	0.12	0	0
6	host4->host2	20GB	No-limited	14.4	21.8	0.3	0.07	4.3	0
7	host1->host3	10GB	No-limited	13.9	19.2	0.01	0.09	0	0

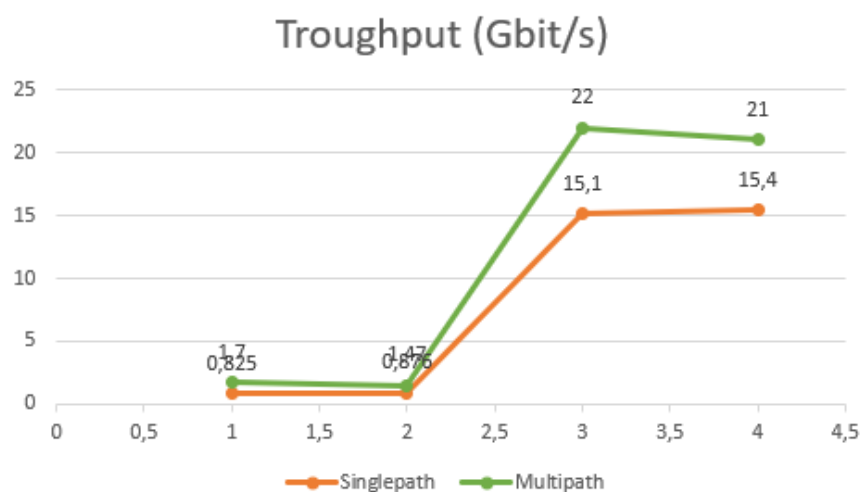
No	Host	Besar Paket	Bandwith	Throughput (Singlepath)/ GBits	Throughput (Multipath)/ GBits	Delay(single path)	Delay (Multipath)	Paket Loss(single path)	Paket Loss(multipath)
8	host2->host3	15GB	No-limited	13.7	21.4	0.11	0.03	0	6.1
9	host4->host3	20GB	No-limited	11.6	20.6	0.21	0.15	0.2	0
10	host1->Host4	10GB	No-limited	15.2	20.3	0.55	0.18	0	0
11	host2->Host4	15GB	No-limited	17.8	21	0.11	0.04	1	0
12	host3->Host4	20GB	No-limited	15.5	20.4	0.2	0.41	2.9	0
TOTAL				15.4	21	0.20	0.09	0.7	0.5

4.3.1 Hasil Perbandingan Pengujian Antara Singlepath dengan Multipath routing

Setelah dilakukannya percobaan pengujian pada *singlepath* dan *Multipath routing* dengan menggunakan skenario 1, skenario 1.2, skenario 2, dan skenario 2.2 maka didapatkan perbandingan antara *singlepath* dan *multipath* dengan melihat *variable* ukur seperti *throughput*, *delay*, dan *paket loss* nilai yang didapatkan pada grafik merupakan hasil pengukuran dari nilai rata-rata tiap pengujian seperti rata-rata nilai pada *throughput singlepath* dan *multipath*.

1) Hasil Throughput

Hasil perbandingan antara *singlepath* dan *multipath routing* dengan melihat metrik *throughput* dapat dilihat pada gambar 15 dibawah.



Gambar 15. Grafik perbandingan *singlepath* dengan *Multipath*

Dari grafik diatas dapat dilihat bahwa setiap percobaan yang sama dilakukan antara *Multipath routing* dan *singlepath routing*, nilai diatas diambil berdasarkan rata-rata setiap skenario yang dilakukan, pada setiap skenario yang dilakukan dapat dilihat bahwa *singlepath* melakukan penurunan yang cukup drastis ketika jalurnya sudah dipakai dan dipakai lagi maka traffic bandwidthnya mengalami penurunan. Pada

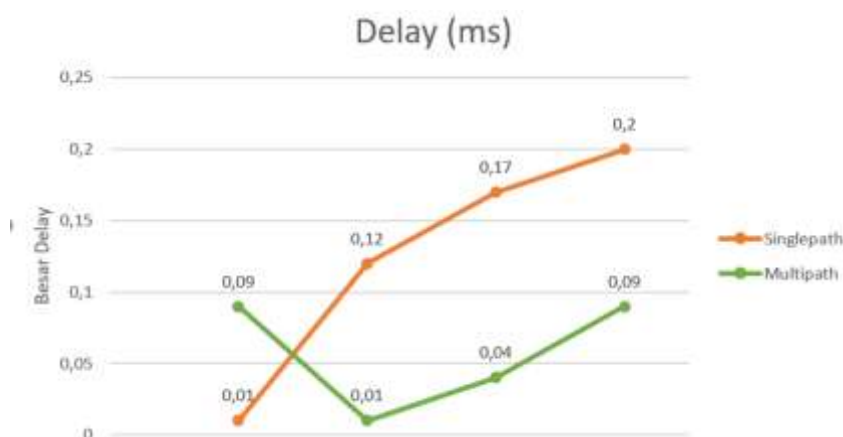
skenario pertama menunjukkan nilai 0,825 Gbit/s pada singlepath dan 1,7 Gbit/s pada multipath hal itu didapatkan karena pada skenario pertama bahwa besar bandwidth pada saat pengiriman dibatasi sebesar 2GB dan hasilnya perbedaan pada singlepath dan multipath tidak terlalu signifikan ,hal itu sama seperti skenario 1.2 karena bandwidth pada pengiriman paket dibatasi dan hasilnya pada skenario 1 dan skenario 1.2 tidak terlalu besar perbedaannya.

Pada skenario 2 dan 2.2 perbedaan throughput antar singlepath dan multipath sudah berbeda jauh karena nilai bandwidth nya dibebaskan atau no-limited bandwidth pada pengiriman paket. Terjadinya perbedaan besar throughput pada singlepath dan multipath, hal itu terjadi karena hanya ada satu path yang dipakai oleh semua *host* untuk mengirimkan paket ke *host* lainnya sehingga sangat tidak memungkinkan untuk memberikan bandwidth lebih banyak lagi karena hanya satu jalur saja yang dipakai. Berbeda dengan *Multipath* ,*Multipath* tetap konsisten dengan besar bandwidth yang dipakai karena banyak jalur yang dipakai *host* untuk mengirimkan paket ke *destination*, jika terjadi traffic yang tinggi pada jalur yang akan dipakai maka masih ada jalur lain yang bisa digunakan untuk melakukan pengiriman paket.

Berdasarkan grafik hasil pengujian diatas, maka dapat diamati bahwa terjadinya peningkatan dalam performa jaringan, mulai dari bandwidth dan throughput hasil yang didapat dari percobaan yang telah dilakukan pada sub bab yang telah dijelaskan diatas. Peningkatan performa jaringan di pengaruhi oleh jalur yang tersedia tidak hanya satu saja sehingga tidak terjadi penumpukan pada satu jalur sehingga sangat rentan terjadinya paket *loss* dan paket *drop*. Dari percobaan yang telah penulis lakukan maka dapat dilihat bahwa penggunaan *Multipath routing* di SDN sangat memberikan peningkatan performa jaringan dalam pengiriman paket dari *host* ke tujuan.

2) Hasil Delay

Hasil perbandingan antara *singlepath* dan *multipath routing* dengan melihat metrik *delay* dapat dilihat pada gambar dibawah 16.



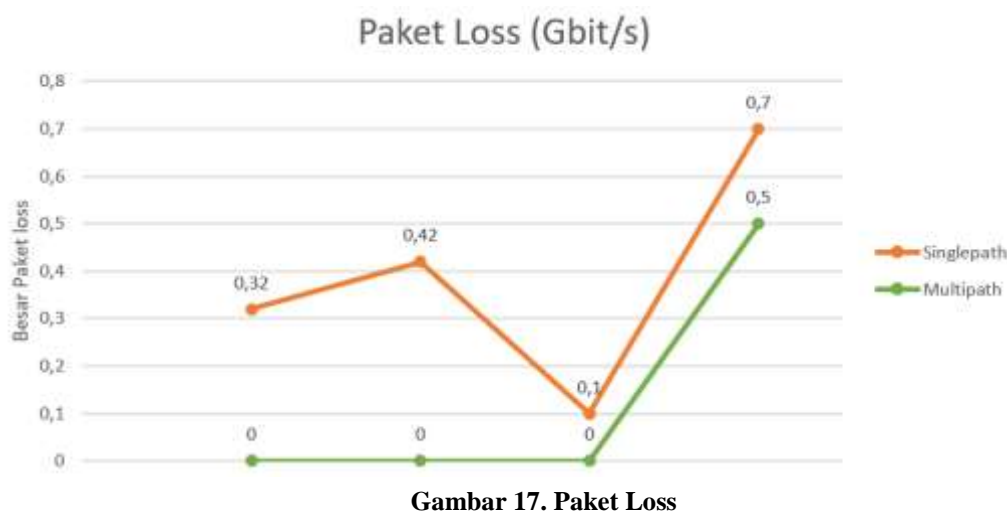
Gambar 16. Host Delay di *Multipath Routing* dan *Singlepath*

Grafik diatas menunjukkan bahwa delay pada singlepath routing lebih tinggi dari pada *multipath routing*. *Singlepath routing* mendapatkan delay yang tinggi dari *multipath routing* dikarenakan penggunaan jalur tunggal sehingga ketika traffik pengiriman paket tinggi dan jumlah paket data yang dikirimkan besar maka dapat mengakibatkan pengiriman paket data *buffering* yang menyebabkan delay yang tinggi.

Namun berbeda pada *multipath routing* yang menggunakan beberapa jalur dalam pengiriman paket data yang mengakibatkan delay ketika pengiriman data lebih rendah. Pada waktu awal pengiriman paket besar delay pada *multipath* lebih tinggi diantara *singlepath* karena pada awal pengiriman paket yang cukup besar membuat jalur yang akan dipakai dalam pengiriman paket tidak dapat mengirimkan paket hanya pada satu jalur saja dan membuat paket akan dipecah-pecah untuk dikirimkan ke semua jalur sehingga pada awal pengiriman delay pada *multipath* menjadi lebih besar, namun setelah pengiriman pertama selesai maka pengiriman selanjutnya delay pada *multipath* akan lebih kecil di bandingkan dengan *singlepath*.

3) Paket Loss

Hasil perbandingan antara *singlepath* dan *multipath routing* dengan melihat *metrik paket loss* dapat dilihat pada gambar 17 dibawah.



Grafik pada gambar 17 menunjukkan bahwa paket loss yang dikirimkan *singlepath routing* lebih tinggi dari *multipath* dikarenakan penggunaan jalur tunggal sehingga pada trafik jaringan yang tinggi maka akan membuat jalur tersebut penuh dan akan memperbesar kemungkinan hilangnya paket yang akan dikirimkan. Sedangkan *multipath* menghasilkan *paket loss* yang rendah karena penggunaan banyak jalur sehingga ketika trafik jaringan tinggi maka dapat dilakukan *load balancing* yang akan menurunkan kemungkinan *paket loss*.

4.4 Perancangan Testbed pada Raspberry Pi

Pada subbab ini akan menjelaskan bagaimana perancangan SDN testbed pada Raspberry Pi, mulai dari instalasi Raspberry Pi, konfigurasi openvswitch dan konfigurasi Ryu sebagai kontroler SDN di testbed pada Raspberry Pi.

4.4.1 Instalasi Raspberry Pi

Raspberry Pi adalah software yang digunakan sebagai *mikroprosesor* untuk mengoperasikan perangkat IoT dan mengirimkan data ke *server*. Berikut akan dijelaskan langkah instalasi sistem operasi Raspbian pada Raspberry Pi 3.

1. Dibutuhkan file installer untuk RPi yaitu NOOBS (New Out of Box Software) yang dapat di download dari <http://www.raspberrypi.org/downloads>.
2. Selanjutnya, dibutuhkan SDFormatter yang dapat di download dari <http://www.sdcard.org/downloads/Formatter.4/> dan Win32DiskImager yang dapat di download di <http://lauchpad.net/win32-image-writer/>. SDFormatter digunakan untuk Format SD card Raspberry Pi, sedangkan Win32DiskImager digunakan untuk write sistem operasi Raspbian ke dalam SD card. Setelah men-download SDFormatter dan Win32DiskImager, masukkan SD card ke SD card reader. Kemudian lakukan format SD Card.
3. Setelah melakukan format pada SD Card, extract NOOBS yang sudah di download ke SD Card.
4. Kemudian pindahkan SD Card ke dalam Raspberry Pi yang akan digunakan, kemudian install Raspbian. 6. SD Card yang sudah berisi sistem operasi Raspbian sudah dapat digunakan langsung untuk perangkat RaspberryPi.

4.4.2 Instalasi OpenVswitch di Raspberry Pi

OpenVswitch merupakan *programmable openflow* yang dapat dikontrol melalui SDN Controller. *OpenVswitch* mendukung pengontrolan secara otomatis dengan protokol *openflow*. *OpenVswitch* dirancang untuk memungkinkan otomatisasi jaringan yang besar melalui ekstensi terprogram, sambil tetap mendukung antarmuka. *OpenVswitch* dapat diinstalasi pada perangkat seperti Raspberry Pi sehingga dapat dikonfigurasi menjadi sebuah topologi fisik sederhana. Berikut akan dijelaskan langkah instalasi *OpenVswitch* pada perangkat Raspberry Pi.

1. Pertama nyalakan perangkat Raspberry Pi kemudian buka terminal dari Raspberry Pi
2. Kemudian *download* file TAR dari *OpenVswitch* dengan perintah

```
pi@raspberrypi~$ sudo wget http://OpenVswitch.org/releases/OpenVswitch-2.7.0.tar.gz
```

3. Unzip folder *OpenVswitch* yang telah terdownload dengan perintah

```
pi@raspberrypi~$ sudo tar -xvf openv
```

4. Pastikan bahwa sudah masuk ke *user privileges* dan install package berikut yang dibutuhkan untuk *running OpenVswitch* pada perangkat Raspberry Pi

```
pi@raspberrypi~$ sudo apt-get install python-simplejson python-qt4 libssl-dev python-twisted-conch automake autoconf gcc uml-utilities libtool build-essential pkg-config
```

5. Cek Kernel Release dari sistem operasi kemudian *copy* dan jalankan command dibawah ini

```
pi@raspberrypi~$ sudo uname -r / apt-cache search linux-headers /apt-get install -y
linux-headers-4.9.0.4-rpi
```

6. Konfigurasi dan *build kernel module* dan kemudian install package

```
pi@raspberrypi~$ sudo ./configure --with-linux=/lib/modules/4.9.0.4-rpi/build
pi@raspberrypi~$ sudo .make
pi@raspberrypi~$ sudo make install
```

7. Kemudian masuk ke direktori datapath/linux yang berada pada folder *OpenVswitch* kemudian jalankan perintah berikut.

```
pi@raspberrypi~$ sudo cd datapath/linux
pi@raspberrypi~# modprobe OpenVswitch
pi@raspberrypi~# cat /etc/modules
pi@raspberrypi~# echo "OpenVswitch" >> /etc/modules
pi@raspberrypi~# cat /etc/modules
```

8. Kemudian kembali ke folder *OpenVswitch* dan *create file ovs database* dengan menjalankan perintah berikut.

```
pi@raspberrypi~$ sudo cd ../../ /touch /usr/local/etc/ovs-vswitchd.conf / mkdir -p
/usr/local/etc/OpenVswitch / ovsdb-tool create /usr/local/etc/OpenVswitch/conf.db
vswitchd/vswitch.ovsschema
```

9. Instalasi *OpenVswitch* juga dapat dilakukan dengan menginstallnya langsung dari package dengan perintah berikut

```
pi@raspberrypi~$ sudo apt-get install OpenVswitch-switch
```

4.4.3 Instalasi Ryu di Raspberry Pi

Ryu merupakan salah satu *Controller* dalam software defined network yang dirancang untuk meningkatkan kemampuan dalam jaringan yang bermanfaat untuk mempermudah dan mengatur skala dalam jaringan serta bebas untuk memberikan algoritma dalam jaringan SDN. RYU merupakan *Controller* yang menggunakan bahasa pemrograman python yang mudah dalam pemakaiannya serta memiliki dokumentasi yang banyak sehingga lebih mudah menemukan solusi jika terdapat permasalahan. Berikut akan dijelaskan langkah instalasi RYU pada perangkat Raspberry Pi.

1. Install ryu menggunakan pip

```
pi@raspberrypi~$ sudo pip install ryu
```

2. Setelah terinstall maka install source code melalui github

```
pi@raspberrypi~$ git clone git://github.com/osrg/ryu.git  
pi@raspberrypi~$ cd ryu; pip install .
```

3. Jika ingin menggunakan fungsi-fungsi ini, silahkan instal persyaratan:

```
pi@raspberrypi~$ sudo pip install -r tools/optional-requirements
```

4. Jika mendapatkan beberapa pesan kesalahan pada tahap instalasi, harap konfirmasi dependensi untuk membangun paket Python yang diperlukan.

```
pi@raspberrypi~$ sudo apt install gcc python-dev libffi-dev libssl-dev libxml2-dev  
libxslt1-dev zlib1g-dev
```


BAB V KESIMPULAN DAN SARAN

5.1 Kesimpulan

Setelah melakukan perancangan dan implementasi *singlepath and multipath routing in terms of network performance and designing SDN testbed* maka didapatkan beberapa kesimpulan yaitu :

1. Penggunaan emulator mininet untuk implementasi SDN menggunakan *multipath routing* dan algoritma ECMP dapat berjalan dengan baik.
2. SDN merupakan konsep jaringan yang baru yang dapat digunakan secara *programable* sehingga mempermudah dalam konfigurasi jaringan yang relatif besar. Implementasi algoritma ECMP pada *multipath routing* di SDN dapat diimplementasikan pada topologi jaringan karena menyediakan *load balancing*.
3. Hasil pengujian antara *multipath routing* dan *singlepath routing* menunjukkan bahwa nilai *throughput multipath routing* lebih tinggi daripada *singlepath routing*, kemudian delay pada *multipath routing* lebih rendah daripada *singlepath routing*, dan *paket loss* pada *multipath routing* lebih rendah daripada *singlepath routing*.
4. Rancangan SDN *testbed* telah berhasil dilakukan sesuai dengan topologi yang digunakan pada *emulator* mininet. Rancangan ini menggunakan Raspberry Pi sebagai *OpenVswitch* dan *Controller*.
5. Hasil pengujian antara *Multipath routing* dan *singlepath routing* dengan membandingkan nilai *metrik* seperti *throuhput*, *delay* dan *paket loss* telah berhasil dilakukan. Kemudian setelah dilakukan perbandingan, maka didapatkan hasil bahwa *multipath routing* di SDN dapat meningkatkan performa jaringan.

5.2 Saran

Setelah dilakukan pengujian pada bab sebelumnya, dapat diketahui bahwa *Multipath routing* dapat meningkatkan performa jaringan dengan memiliki banyak jalur untuk dapat melakukan pengiriman data atau paket dengan menggunakan algoritma. Namun pada SDN *testbed*, penulis belum dapat mengimplementasikan rancangan sesuai dengan mininet karena penulis belum bisa untuk melakukan konfigurasi antar kontroler dan *OpenVswitch* dalam melakukan percobaan. Untuk kedepannya diharapkan dapat dilakukan pengembangan untuk pengimplementasian *Raspberry Pi* sebagai *OpenVswitch* dan *Controller* sebagai implementasi SDN *testbed*.

Daftar Pustaka

- [1] H. Abrahamsson, B. Ahlgren, J. Alonso, A. Andersson, and P. Kreuger, "A multi-path routing algorithm for IP networks based on flow optimisation," From QoS Provisioning to QoS Chargin, pp. 135–144, 2002.
- [2] Mukti, Fransiska Sisilia., Basuki, A., dkk., 2018. "Pengendalian Kemacetan Jaringan Melalui Per-Flow Multipath Routing "
- [3] Ramadhona Nurul., 2017. "Pengenalan Teknologi SDN (*Software Defined Networking*)"
- [4] F. Ieee et al., "Software-Defined Networking : A Comprehensive Survey," Proc. IEEE, vol. 103, no. 1, pp. 14–76, 2015.
- [2] R. Singh and A. Yadav, "Loop Free Multipath Routing Algorithm,"arXiv:1601.01245, 2016.
- [3] S. Sharma, D. Staessens, D. Colle, M. Pickavet and P. Demeester, "Enabling Fast Failure Recovery in Openflow Networks," in 8th Internasional Workshop on the DRCN, 2011.
- [4] McKeown Nick, "Openflow: Enabling Innovation in Campus Networks", ACM SIGCOMM, 38, 2, 2007.
- [5] Kim, H. & Feamster, N., 2013. "Improving Network Management with Software Defined Networking. IEEE Communications Magazine, "51(2), pp. 114 – 119.
- [6]Maulana, W., 2017. "Multipath Routing dengan Load-Balancing Pada Openflow Software-Defined Network." Repositori Jurnal Mahasiswa PTIIK UB, Volume 9, p. 15
- [7] Nunes, B. A. A. et al., 2014. "A Survey of Software-Defined Networking: Past,Present, and Future of Programmable Networks. IEEE Communications Surveys & Tutorials", 16(3), pp. 1617 - 1634.
- [8] K. T. Dinh, S. Kuklinski, W. Kujawa, M. Ulaski. "MSDN-TE: Multipath Based Traffic Engineering for SDN," in Proc. 8th Asian Conference, ACIIDS 2016, pp. 630-639, 2016.p.58–69, 2015.
- [9] A. Lappetel, "Equal Cost Multipath Routing in IP Networks," 2011.
- [10] C. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm," Doc. RFC 2992, IETF, pp. 1–8, 2000.
- [11] Negara Muldina R. , Tulloh Rohmat , "Analisis Simulasi Penerapan Algoritma OSPF Menggunakan RouteFlowpada Jaringan Software Defined Network(SDN) "

- [12] Gopakumar, R., Unni, A. M., & Dhipin, V. P. (2015). “*An adaptive algorithm for searching in flow tables of openflow switches*”. 2015 39th National Systems Conference (NSC).
- [13] The Linux Foundation, “Openvswitch.org.” <http://openvswitch.org/>. Accessed August, 2019.
- [14] Getting Started – Raspberry Pi documentation. [Online].
<https://www.raspberrypi.org>
- [15] Getting Started - Ryu 4.2.4 documentation. [Online].
https://ryu.readthedocs.io/en/latest/getting_started.html#what-s-ryu
- [16] Luis Guillen, Satoru Izumi (2015),”*SDN Implementation of Multipath Discovery to Improve Network Performance in Distributed Storage Systems*”. 978-3-901882-98-2, 2015.
- [17] A. Tirumala, M. Gates, F. Qin, J. Dugan and J. Ferguson. “Iperf - The TCP/UDP bandwidth measurement tool”. [Online]. Available: <http://dast.nlanr.net/Projects/Iperf>
- [18] Mininet Team, “Mininet: An instant virtual network on your laptop (or other pc) - mininet.” <http://mininet.org>. Accessed August 2019.
- [19] Dewo E Setio, “*Bandwidth dan Throughput*”, 2003
- [20] Joe, E., Pan, D., Liu, J. & Butler, L., 2014. “*A Simulation and Emulation Study of SDN Based Multipath Routing for Fat-Tree Data Center Networks. Proceedings of the*”, 2014 Winter Simulation Conference, IEEE.
- [21] Dewanto, R., Munadi, R., & Muldina, R., 2018. “Minimalisasi Pemilihan Rute Overlap Pada Equal Cost Multipath Routing (ecmp) Dengan Pendekatan Software Defined Networking”
- [22] Jinyao, Y., Hailong, Z., dkk., 2015. “*HiQoS: An SDN-based Multipath QoS solution*”
- [23] Fan, Ziyang ,2018 . “Multipath Routing in SDN for Network Performance Improvement”
- [24] J. Postel, "Internet Control Message Protocol", STD 5, RFC 792, USC/Information Sciences Institute, September 2012.

LAMPIRAN

1. Kode program untuk *Singlepath routing*

Single_path.py

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.link import TCLink, Link

class Topologi(Topo):

    def __init__(self):

        Topo.__init__(self)

        h1 = self.addHost('h1', mac='00:00:00:00:01:01', ip='10.0.0.1/24')
        h2 = self.addHost('h2', mac='00:00:00:00:01:02', ip='10.0.0.2/24')
        h3 = self.addHost('h3', mac='00:00:00:00:01:03', ip='10.0.0.3/24')
        h4 = self.addHost('h4', mac='00:00:00:00:01:04', ip='10.0.0.4/24')

        s1 = self.addSwitch('s1', mac='00:00:00:00:00:01')
        s2 = self.addSwitch('s2', mac='00:00:00:00:00:02')
        s3 = self.addSwitch('s3', mac='00:00:00:00:00:03')
        s4 = self.addSwitch('s4', mac='00:00:00:00:00:04')

        self.addLink(s1, h1)
        self.addLink(s2, h2)
        self.addLink(s3, h3)
        self.addLink(s4, h4)

        self.addLink(s1, s2, cls=TCLink, delay='1ms', loss=0, max_queue_size=1000, use_htb=True)
        self.addLink(s2, s3, cls=TCLink, delay='4ms', loss=0, max_queue_size=1000, use_htb=True)
        self.addLink(s3, s4, cls=TCLink, delay='2ms', loss=0, max_queue_size=1000, use_htb=True)

        topos = {'switch4': (lambda: Topologi())}
```

2. Kode program untuk *Multipathpath routing*

Multipath_path.py

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.link import TCLink, Link

class Topologi(Topo):

    def __init__(self):

        Topo.__init__(self)

        h1 = self.addHost('h1', mac='00:00:00:00:01:01', ip='10.0.0.1/24')
        h2 = self.addHost('h2', mac='00:00:00:00:01:02', ip='10.0.0.2/24')
        h3 = self.addHost('h3', mac='00:00:00:00:01:03', ip='10.0.0.3/24')
```

```

h4 = self.addHost('h4', mac='00:00:00:00:01:04',ip='10.0.0.4/24')

s1 = self.addSwitch('s1',mac='00:00:00:00:00:01')
s2 = self.addSwitch('s2',mac='00:00:00:00:00:02')
s3 = self.addSwitch('s3',mac='00:00:00:00:00:03')
s4 = self.addSwitch('s4',mac='00:00:00:00:00:04')

self.addLink(s1,h1)
self.addLink(s2,h2)
self.addLink(s3,h3)
self.addLink(s4,h4)

self.addLink(s1,s2,cls=TCLink, delay='1ms',loss=0,max_queue_size=1000,use_htb=True)
self.addLink(s2,s3,cls=TCLink,delay='1ms',loss=0,max_queue_size=1000,use_htb=True)
self.addLink(s3,s4,cls=TCLink, delay='1ms',loss=0,max_queue_size=1000,use_htb=True)
self.addLink(s4,s1,cls=TCLink,delay='1ms',loss=0,max_queue_size=1000,use_htb=True)
self.addLink(s1,s3,cls=TCLink, delay='1ms',loss=0,max_queue_size=1000,use_htb=True)
self.addLink(s2,s4,cls=TCLink,delay='1ms',loss=0,max_queue_size=1000,use_htb=True)

topos = {'switch4':(lambda:Topologi())}

```

3. Kode program untuk algoritma ECMP

ECMP_.py

```

from mininet.net import Mininet
from mininet.node import Controller, RemoteController
from mininet.cli import CLI
from mininet.log import setLogLevel
from mininet.link import Link, Intf, TCLink
from mininet.topo import Topo
from mininet.util import dumpNodeConnections

import logging
import os

class Fattree(Topo):
    """
    Class of Fattree Topology.
    """
    CoreSwitchList = []
    AggSwitchList = []
    EdgeSwitchList = []
    HostList = []

    def __init__(self, k, density):
        self.pod = k
        self.density = density
        self.iCoreLayerSwitch = (k/2)**2
        self.iAggLayerSwitch = k*k/2
        self.iEdgeLayerSwitch = k*k/2

```

```

self.iHost = self.iEdgeLayerSwitch * density

# Init Topo
Topo.__init__(self)

def createNodes(self):
    self.createCoreLayerSwitch(self.iCoreLayerSwitch)
    self.createAggLayerSwitch(self.iAggLayerSwitch)
    self.createEdgeLayerSwitch(self.iEdgeLayerSwitch)
    self.createHost(self.iHost)

# Create Switch and Host
def _addSwitch(self, number, level, switch_list):
    """
        Create switches.
    """
    for i in xrange(1, number+1):
        PREFIX = str(level) + "00"
        if i >= 10:
            PREFIX = str(level) + "0"
        switch_list.append(self.addSwitch(PREFIX + str(i)))

def createCoreLayerSwitch(self, NUMBER):
    self._addSwitch(NUMBER, 1, self.CoreSwitchList)

def createAggLayerSwitch(self, NUMBER):
    self._addSwitch(NUMBER, 2, self.AggSwitchList)

def createEdgeLayerSwitch(self, NUMBER):
    self._addSwitch(NUMBER, 3, self.EdgeSwitchList)

def createHost(self, NUMBER):
    """
        Create hosts.
    """
    for i in xrange(1, NUMBER+1):
        if i >= 100:
            PREFIX = "h"
        elif i >= 10:
            PREFIX = "h0"
        else:
            PREFIX = "h00"
        self.HostList.append(self.addHost(PREFIX + str(i), cpu=1.0/NUMBER))

def createLinks(self, bw_c2a=10, bw_a2e=10, bw_e2h=10):
    """
        Add network links.
    """
    # Core to Agg
    end = self.pod/2
    for x in xrange(0, self.iAggLayerSwitch, end):
        for i in xrange(0, end):
            for j in xrange(0, end):
                self.addLink(

```

```

self.CoreSwitchList[i*end+j],
self.AggSwitchList[x+i],
bw=bw_c2a, max_queue_size=1000) #
use_htb=False

    # Agg to Edge
    for x in xrange(0, self.iAggLayerSwitch, end):
        for i in xrange(0, end):
            for j in xrange(0, end):
                self.addLink(
                    self.AggSwitchList[x+i],
self.EdgeSwitchList[x+j],

                    bw=bw_a2e, max_queue_size=1000) #
use_htb=False

    # Edge to Host
    for x in xrange(0, self.iEdgeLayerSwitch):
        for i in xrange(0, self.density):
            self.addLink(
                self.EdgeSwitchList[x],
                self.HostList[self.density * x + i],
                bw=bw_e2h, max_queue_size=1000) # use_htb=False

def set_ovs_protocol_13(self,):
    """
        Set the Openflow version for switches.
    """
    self._set_ovs_protocol_13(self.CoreSwitchList)
    self._set_ovs_protocol_13(self.AggSwitchList)
    self._set_ovs_protocol_13(self.EdgeSwitchList)

def _set_ovs_protocol_13(self, sw_list):
    for sw in sw_list:
        cmd = "sudo ovs-vsctl set bridge %s protocols=Openflow13" % sw
        os.system(cmd)

def set_host_ip(net, topo):
    hostlist = []
    for k in xrange(len(topo.HostList)):
        hostlist.append(net.get(topo.HostList[k]))
    i = 1
    j = 1
    for host in hostlist:
        host.setIP("10.%d.0.%d" % (i, j))
        j += 1
        if j == topo.density+1:
            j = 1
            i += 1

def create_subnetList(topo, num):
    """
        Create the subnet list of the certain Pod.
    """

```

```

subnetList = []
remainder = num % (topo.pod/2)
if topo.pod == 4:
    if remainder == 0:
        subnetList = [num-1, num]
    elif remainder == 1:
        subnetList = [num, num+1]
    else:
        pass
elif topo.pod == 8:
    if remainder == 0:
        subnetList = [num-3, num-2, num-1, num]
    elif remainder == 1:
        subnetList = [num, num+1, num+2, num+3]
    elif remainder == 2:
        subnetList = [num-1, num, num+1, num+2]
    elif remainder == 3:
        subnetList = [num-2, num-1, num, num+1]
    else:
        pass
else:
    pass
return subnetList

def install_proactive(net, topo):
    """
    Install proactive flow entries for switches.
    """
    # Edge Switch
    for sw in topo.EdgeSwitchList:
        num = int(sw[-2:])

        # Downstream.
        for i in xrange(1, topo.density+1):
            cmd = "ovs-ofctl add-flow %s -O Openflow13 \
                'table=0,idle_timeout=0,hard_timeout=0,priority=40,arp, \
                nw_dst=10.%d.0.%d,actions=output:%d'" % (sw, num, i,
topo.pod/2+i)
            os.system(cmd)
            cmd = "ovs-ofctl add-flow %s -O Openflow13 \
                'table=0,idle_timeout=0,hard_timeout=0,priority=40,ip, \
                nw_dst=10.%d.0.%d,actions=output:%d'" % (sw, num, i,
topo.pod/2+i)
            os.system(cmd)

        # Upstream.
        if topo.pod == 4:
            cmd = "ovs-ofctl add-group %s -O Openflow13 \
                'group_id=1,type=select,bucket=output:1,bucket=output:2'" % sw
        elif topo.pod == 8:
            cmd = "ovs-ofctl add-group %s -O Openflow13 \
                'group_id=1,type=select,bucket=output:1,bucket=output:2,\
                bucket=output:3,bucket=output:4'" % sw
        else:

```



```

        pass
    os.system(cmd)
    cmd = "ovs-ofctl add-flow %s -O Openflow13 \
'table=0,priority=10,arp,actions=group:1'" % sw
    os.system(cmd)
    cmd = "ovs-ofctl add-flow %s -O Openflow13 \
'table=0,priority=10,ip,actions=group:1'" % sw
    os.system(cmd)

# Aggregate Switch
for sw in topo.AggSwitchList:
    num = int(sw[-2:])
    subnetList = create_subnetList(topo, num)

# Downstream.
k = 1
for i in subnetList:
    cmd = "ovs-ofctl add-flow %s -O Openflow13 \
'table=0,idle_timeout=0,hard_timeout=0,priority=40,arp, \
nw_dst=10.%d.0.0/16, actions=output:%d'" % (sw, i,
topo.pod/2+k)

    os.system(cmd)
    cmd = "ovs-ofctl add-flow %s -O Openflow13 \
'table=0,idle_timeout=0,hard_timeout=0,priority=40,ip, \
nw_dst=10.%d.0.0/16, actions=output:%d'" % (sw, i,
topo.pod/2+k)

    os.system(cmd)
    k += 1

# Upstream.
if topo.pod == 4:
    cmd = "ovs-ofctl add-group %s -O Openflow13 \
'group_id=1,type=select,bucket=output:1,bucket=output:2'" % sw
elif topo.pod == 8:
    cmd = "ovs-ofctl add-group %s -O Openflow13 \
'group_id=1,type=select,bucket=output:1,bucket=output:2,\
bucket=output:3,bucket=output:4'" % sw
else:
    pass
os.system(cmd)
cmd = "ovs-ofctl add-flow %s -O Openflow13 \
'table=0,priority=10,arp,actions=group:1'" % sw
os.system(cmd)
cmd = "ovs-ofctl add-flow %s -O Openflow13 \
'table=0,priority=10,ip,actions=group:1'" % sw
os.system(cmd)

# Core Switch
for sw in topo.CoreSwitchList:
    j = 1
    k = 1
    for i in xrange(1, len(topo.EdgeSwitchList)+1):
        cmd = "ovs-ofctl add-flow %s -O Openflow13 \
'table=0,idle_timeout=0,hard_timeout=0,priority=10,arp, \

```

```

        nw_dst=10.%d.0.0/16, actions=output:%d" % (sw, i, j)
    os.system(cmd)
    cmd = "ovs-ofctl add-flow %s -O Openflow13 \
        'table=0,idle_timeout=0,hard_timeout=0,priority=10,ip, \
        nw_dst=10.%d.0.0/16, actions=output:%d" % (sw, i, j)
    os.system(cmd)
    k += 1
    if k == topo.pod/2 + 1:
        j += 1
        k = 1

def iperfTest(net, topo):
    """
        Start iperf test.
    """
    h001, h015, h016 = net.get(
        topo.HostList[0], topo.HostList[14], topo.HostList[15])
    # iperf Server
    h001.popen('iperf -s -u -i 1 > iperf_server_differentPod_result', shell=True)
    # iperf Server
    h015.popen('iperf -s -u -i 1 > iperf_server_samePod_result', shell=True)
    # iperf Client
    h016.cmdPrint('iperf -c ' + h001.IP() + ' -u -t 10 -i 1 -b 10m')
    h016.cmdPrint('iperf -c ' + h015.IP() + ' -u -t 10 -i 1 -b 10m')

def pingTest(net):
    """
        Start ping test.
    """
    net.pingAll()

def createTopo(pod, density, ip="192.168.56.101", port=6653, bw_c2a=10, bw_a2e=10,
    bw_e2h=10):
    """
        Create network topology and run the Mininet.
    """
    # Create Topo.
    topo = Fattree(pod, density)
    topo.createNodes()
    topo.createLinks(bw_c2a=bw_c2a, bw_a2e=bw_a2e, bw_e2h=bw_e2h)

    # Start Mininet.
    CONTROLLER_IP = ip
    CONTROLLER_PORT = port
    net = Mininet(topo=topo, link=TCLink, Controller=None, autoSetMacs=True)
    net.addController(
        'Controller', Controller=RemoteController,
        ip=CONTROLLER_IP, port=CONTROLLER_PORT)
    net.start()

    # Set OVS's protocol as OF13.
    topo.set_ovs_protocol_13()
    # Set hosts IP addresses.
    set_host_ip(net, topo)

```

```

        # Install proactive flow entries
        install_proactive(net, topo)
        # dumpNodeConnections(net.hosts)
        # pingTest(net)
        # iperfTest(net, topo)

        CLI(net)
        net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    if os.getuid() != 0:
        logging.debug("You are NOT root")
    elif os.getuid() == 0:
        createTopo(4, 2)
        # createTopo(8, 4)

```

4. Kode program Algoritma DFS

DFS.py

```

from ryu.base import app_manager
from ryu.Controller import mac_to_port
from ryu.Controller import ofp_event
from ryu.Controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.Controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.mac import haddr_to_bin
from ryu.lib.packet import packet
from ryu.lib.packet import arp
from ryu.lib.packet import ethernet
from ryu.lib.packet import ipv4
from ryu.lib.packet import ipv6
from ryu.lib.packet import ether_types
from ryu.lib import mac, ip
from ryu.topology.api import get_switch, get_link
from ryu.app.wsgi import ControllerBase
from ryu.topology import event

from collections import defaultdict
from operator import itemgetter

import os
import random
import time

# bandwidth = 1 Gbps
REFERENCE_BW = 10000000

DEFAULT_BW = 10000000

MAX_PATHS = 2

```

```

class ProjectController(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(ProjectController, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
        self.topology_api_app = self
        self.datapath_list = {}
        self.arp_table = {}
        self.switches = []
        self.hosts = {}
        self.Multipath_group_ids = {}
        self.group_ids = []
        self.adjacency = defaultdict(dict)
        self.bandwidths = defaultdict(lambda: defaultdict(lambda: DEFAULT_BW))

    def get_paths(self, src, dst):
        """
        Get all paths from src to dst using DFS algorithm
        """
        if src == dst:
            # host target is on the same switch
            return [[src]]
        paths = []
        stack = [(src, [src])]
        while stack:
            (node, path) = stack.pop()
            for next in set(self.adjacency[node].keys()) - set(path):
                if next is dst:
                    paths.append(path + [next])
                else:
                    stack.append((next, path + [next]))
        print "Available paths from ", src, " to ", dst, " : ", paths
        return paths

    def get_link_cost(self, s1, s2):
        """
        Get the link cost between two switches
        """
        e1 = self.adjacency[s1][s2]
        e2 = self.adjacency[s2][s1]
        b1 = min(self.bandwidths[s1][e1], self.bandwidths[s2][e2])
        ew = REFERENCE_BW/b1
        return ew

    def get_path_cost(self, path):
        """
        Get the path cost
        """
        cost = 0
        for i in range(len(path) - 1):
            cost += self.get_link_cost(path[i], path[i+1])
        return cost

```

```

def get_optimal_paths(self, src, dst):
    """
    Get the n-most optimal paths according to MAX_PATHS
    """
    paths = self.get_paths(src, dst)
    paths_count = len(paths) if len(
        paths) < MAX_PATHS else MAX_PATHS
    return sorted(paths, key=lambda x: self.get_path_cost(x))[0:(paths_count)]

def add_ports_to_paths(self, paths, first_port, last_port):
    """
    Add the ports that connects the switches for all paths
    """
    paths_p = []
    for path in paths:
        p = {}
        in_port = first_port
        for s1, s2 in zip(path[:-1], path[1:]):
            out_port = self.adjacency[s1][s2]
            p[s1] = (in_port, out_port)
            in_port = self.adjacency[s2][s1]
        p[path[-1]] = (in_port, last_port)
        paths_p.append(p)
    return paths_p

def generate_openflow_gid(self):
    """
    Returns a random Openflow group id
    """
    n = random.randint(0, 2**32)
    while n in self.group_ids:
        n = random.randint(0, 2**32)
    return n

def install_paths(self, src, first_port, dst, last_port, ip_src, ip_dst):
    computation_start = time.time()
    paths = self.get_optimal_paths(src, dst)
    pw = []
    for path in paths:
        pw.append(self.get_path_cost(path))
        print path, "cost = ", pw[len(pw) - 1]
    sum_of_pw = sum(pw) * 1.0
    paths_with_ports = self.add_ports_to_paths(paths, first_port, last_port)
    switches_in_paths = set().union(*paths)

    for node in switches_in_paths:
        dp = self.datapath_list[node]
        ofp = dp.ofproto
        ofp_parser = dp.ofproto_parser

        ports = defaultdict(list)
        actions = []

```

```

i = 0

for path in paths_with_ports:
    if node in path:
        in_port = path[node][0]
        out_port = path[node][1]
        if (out_port, pw[i]) not in ports[in_port]:
            ports[in_port].append((out_port, pw[i]))
        i += 1

for in_port in ports:

    match_ip = ofp_parser.OFPMatch(
        eth_type=0x0800,
        ipv4_src=ip_src,
        ipv4_dst=ip_dst
    )
    match_arp = ofp_parser.OFPMatch(
        eth_type=0x0806,
        arp_spa=ip_src,
        arp_tpa=ip_dst
    )

    out_ports = ports[in_port]
    # print out_ports

    if len(out_ports) > 1:
        group_id = None
        group_new = False

        if (node, src, dst) not in self.Multipath_group_ids:
            group_new = True
            self.Multipath_group_ids[
                node, src, dst] = self.generate_openflow_gid()
            group_id = self.Multipath_group_ids[node, src, dst]

        buckets = []
        # print "node at ",node," out ports : ",out_ports
        for port, weight in out_ports:
            bucket_weight = int(round((1 - weight/sum_of_pw) * 10))
            bucket_action = [ofp_parser.OFPActionOutput(port)]
            buckets.append(
                ofp_parser.OFPBucket(
                    weight=bucket_weight,
                    watch_port=port,
                    watch_group=ofp.OFPG_ANY,
                    actions=bucket_action
                )
            )

        if group_new:
            req = ofp_parser.OFPGGroupMod(
                dp, ofp.OFPGC_ADD, ofp.OFPGT_SELECT, group_id,
                buckets

```

```

        )
        dp.send_msg(req)
    else:
        req = ofp_parser.OFPGroupMod(
            dp, ofp.OFPGC_MODIFY, ofp.OFPGT_SELECT,
            group_id, buckets)
        dp.send_msg(req)

    actions = [ofp_parser.OFPActionGroup(group_id)]

    self.add_flow(dp, 32768, match_ip, actions)
    self.add_flow(dp, 1, match_arp, actions)

    elif len(out_ports) == 1:
        actions = [ofp_parser.OFPActionOutput(out_ports[0][0])]

        self.add_flow(dp, 32768, match_ip, actions)
        self.add_flow(dp, 1, match_arp, actions)
    print "Path installation finished in ", time.time() - computation_start
    return paths_with_ports[0][src][1]

def add_flow(self, datapath, priority, match, actions, buffer_id=None):
    # print "Adding flow ", match, actions
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                         actions)]

    if buffer_id:
        mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                priority=priority, match=match,
                                instructions=inst)
    else:
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                match=match, instructions=inst)
    datapath.send_msg(mod)

@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def _switch_features_handler(self, ev):
    print "switch_features_handler is called"
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    match = parser.OFPMatch()
    actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                       ofproto.OFPCML_NO_BUFFER)]
    self.add_flow(datapath, 0, match, actions)

@set_ev_cls(ofp_event.EventOFPPortDescStatsReply, MAIN_DISPATCHER)
def port_desc_stats_reply_handler(self, ev):
    switch = ev.msg.datapath
    for p in ev.msg.body:
        self.bandwidths[switch.id][p.port_no] = p.curr_speed

```

```

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']

    pkt = packet.Packet(msg.data)
    eth = pkt.get_protocol(ethernet.ethernet)
    arp_pkt = pkt.get_protocol(arp.arp)

    # avoid broadcast from LLDP
    if eth.ethertype == 35020:
        return

    if pkt.get_protocol(ipv6.ipv6): # Drop the IPV6 Packets.
        match = parser.OFPMatch(eth_type=eth.ethertype)
        actions = []
        self.add_flow(datapath, 1, match, actions)
        return None

    dst = eth.dst
    src = eth.src
    dpid = datapath.id

    if src not in self.hosts:
        self.hosts[src] = (dpid, in_port)

    out_port = ofproto.OFPP_FLOOD

    if arp_pkt:
        # print dpid, pkt
        src_ip = arp_pkt.src_ip
        dst_ip = arp_pkt.dst_ip
        if arp_pkt.opcode == arp.ARP_REPLY:
            self.arp_table[src_ip] = src
            h1 = self.hosts[src]
            h2 = self.hosts[dst]
            out_port = self.install_paths(h1[0], h1[1], h2[0], h2[1], src_ip, dst_ip)
            self.install_paths(h2[0], h2[1], h1[0], h1[1], dst_ip, src_ip) # reverse
        elif arp_pkt.opcode == arp.ARP_REQUEST:
            if dst_ip in self.arp_table:
                self.arp_table[src_ip] = src
                dst_mac = self.arp_table[dst_ip]
                h1 = self.hosts[src]
                h2 = self.hosts[dst_mac]
                out_port = self.install_paths(h1[0], h1[1], h2[0], h2[1], src_ip, dst_ip)
                self.install_paths(h2[0], h2[1], h1[0], h1[1], dst_ip, src_ip) # reverse

    # print pkt

    actions = [parser.OFPACTIONOutput(out_port)]

```



```

data = None
if msg.buffer_id == ofproto.OFP_NO_BUFFER:
    data = msg.data

out = parser.OFPPacketOut(
    datapath=datapath, buffer_id=msg.buffer_id, in_port=in_port,
    actions=actions, data=data)
datapath.send_msg(out)

@set_ev_cls(event.EventSwitchEnter)
def switch_enter_handler(self, ev):
    switch = ev.switch.dp
    ofp_parser = switch.ofproto_parser

    if switch.id not in self.switches:
        self.switches.append(switch.id)
        self.datapath_list[switch.id] = switch

    # Request port/link descriptions, useful for obtaining bandwidth
    req = ofp_parser.OFPPortDescStatsRequest(switch)
    switch.send_msg(req)

@set_ev_cls(event.EventSwitchLeave, MAIN_DISPATCHER)
def switch_leave_handler(self, ev):
    print ev
    switch = ev.switch.dp.id
    if switch in self.switches:
        self.switches.remove(switch)
        del self.datapath_list[switch]
        del self.adjacency[switch]

@set_ev_cls(event.EventLinkAdd, MAIN_DISPATCHER)
def link_add_handler(self, ev):
    s1 = ev.link.src
    s2 = ev.link.dst
    self.adjacency[s1.dpid][s2.dpid] = s1.port_no
    self.adjacency[s2.dpid][s1.dpid] = s2.port_no

@set_ev_cls(event.EventLinkDelete, MAIN_DISPATCHER)
def link_delete_handler(self, ev):
    s1 = ev.link.src
    s2 = ev.link.dst
    # Exception handling if switch already deleted
    try:
        del self.adjacency[s1.dpid][s2.dpid]
        del self.adjacency[s2.dpid][s1.dpid]
    except KeyError:
        pass

```

5. Kode program untuk Algoritma OSPF

shortestpath13.py

```
from ryu.base import app_manager
from ryu.Controller import mac_to_port
from ryu.Controller import ofp_event
from ryu.Controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.Controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.mac import haddr_to_bin
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types
from ryu.lib import mac
from ryu.topology.api import get_switch, get_link
from ryu.app.wsgi import ControllerBase
from ryu.topology import event, switches
import networkx as nx
from shortestpath13

class ProjectController(app_manager.RyuApp):

    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):

        super(ProjectController, self).__init__(*args, **kwargs)

        self.mac_to_port = {}

        self.topology_api_app = self

        self.net=nx.DiGraph()

        self.nodes = {}

        self.links = {}

        self.no_of_nodes = 0

        self.no_of_links = 0

        self.i=0

        # Handy function that lists all attributes in the given object
```

```

def ls(self,obj):

    print("\n".join([x for x in dir(obj) if x[0] != "_"]))


def add_flow(self, datapath, in_port, dst, actions):

    ofproto = datapath.ofproto

    parser = datapath.ofproto_parser

    match = datapath.ofproto_parser.OFPMatch(in_port=in_port, eth_dst=dst)

    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]

    mod = datapath.ofproto_parser.OFPFlowMod(

        datapath=datapath, match=match, cookie=0,

        command=ofproto.OFPFC_ADD, idle_timeout=0, hard_timeout=0,

        priority=ofproto.OFP_DEFAULT_PRIORITY, instructions=inst)

    datapath.send_msg(mod)


@set_ev_cls(ofp_event.EventOFPSwitchFeatures , CONFIG_DISPATCHER)

def switch_features_handler(self , ev):

    print "switch_features_handler is called"

    datapath = ev.msg.datapath

    ofproto = datapath.ofproto

    parser = datapath.ofproto_parser

    match = parser.OFPMatch()

    actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
ofproto.OFPCML_NO_BUFFER)]

    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS , actions)]

    mod = datapath.ofproto_parser.OFPFlowMod(

        datapath=datapath, match=match, cookie=0,

        command=ofproto.OFPFC_ADD, idle_timeout=0, hard_timeout=0, priority=0,
instructions=inst)

```

```

datapath.send_msg(mod)

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    msg = ev.msg

    datapath = msg.datapath

    ofproto = datapath.ofproto

    parser = datapath.ofproto_parser

    in_port = msg.match['in_port']

    pkt = packet.Packet(msg.data)

    eth = pkt.get_protocol(ethernet.ethernet)

    dst = eth.dst

    src = eth.src

    dpid = datapath.id

    self.mac_to_port.setdefault(dpid, { })

    #print "nodes"

    #print self.net.nodes()

    #print "edges"

    #print self.net.edges()

    #self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

    if src not in self.net:

        self.net.add_node(src)

        self.net.add_edge(dpid,src,{'port':in_port})

        self.net.add_edge(src,dpid)

```

```

if dst in self.net:

    #print (src in self.net)

    #print nx.shortest_path(self.net,1,4)

    #print nx.shortest_path(self.net,4,1)

    #print nx.shortest_path(self.net,src,4)


    path=nx.shortest_path(self.net,src,dst)

    next=path[path.index(dpid)+1]

    out_port=self.net[dpid][next]['port']

else:

    out_port = ofproto.OFPP_FLOOD


actions = [datapath.ofproto_parser.OFPActionOutput(out_port)]

# install a flow to avoid packet_in next time

if out_port != ofproto.OFPP_FLOOD:

    self.add_flow(datapath, in_port, dst, actions)


out = datapath.ofproto_parser.OFPPacketOut(

    datapath=datapath, buffer_id=msg.buffer_id, in_port=in_port,

    actions=actions)

datapath.send_msg(out)


@set_ev_cls(event.EventSwitchEnter)

def get_topology_data(self, ev):

    switch_list = get_switch(self.topology_api_app, None)

    switches=[switch.dp.id for switch in switch_list]

    self.net.add_nodes_from(switches)

```

```

print "*****List of switches"

for switch in switch_list:

    #self.ls(switch)

    print switch

    #self.nodes[self.no_of_nodes] = switch

    #self.no_of_nodes += 1

links_list = get_link(self.topology_api_app, None)

#print links_list

links=[(link.src.dpid,link.dst.dpid,{ 'port':link.src.port_no}) for link in links_list]

#print links

self.net.add_edges_from(links)

links=[(link.dst.dpid,link.src.dpid,{ 'port':link.dst.port_no}) for link in links_list]

#print links

self.net.add_edges_from(links)

print "*****List of links"

print self.net.edges()

```