ANALISIS PERBANDINGAN KINERJA ALGORITME *DIJKSTRA, BELLMAN-FORD* DAN *FLOYD-WARSHALL* UNTUK PENENTUAN JALUR TERPENDEK PADA ARSITEKTUR JARINGAN *SOFTWARE DEFINED NETWORK*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Aprillia Arum Pratiwi
NIM: 135150201111200

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

ANALISIS PERBANDINGAN KINERJA ALGORITME *DIJKSTRA, BELLMAN-FORD, DAN FLOYD-WARSHALL* UNTUK PENENTUAN JALUR TERPENDEK PADA ARSITEKTUR JARINGAN *SOFTWARE DEFINED NETWORK*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Aprillia Arum Pratiwi
NIM: 135150201111200

Skripsi ini telah diuji dan dinyatakan lulus pada
2 Agustus 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Widhi Yahya, S.Kom., M.Sc.	Mahendra Data, S.Kom., M.Kom
NIK. 201607 891121 1001	NIK. 201503 861117 1001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T., M.T., Ph.D
NIP. 19710518 200312 1 001

IDENTITAS TIM PENGUJI

Majelis Penguji Ujian Skripsi

1. Fariz Andri Bakhtiar, S.T., M.Kom. (ke I)
   NIK. 2017098403141001
   (Bertindak sebagai Ketua Majelis)
2. Eko Sakti Pramukantoro, S.Kom, M.Kom (ke II)
   NIK. 201102 860805 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata di dalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 2 Agustus 2018

Aprillia Arum Pratiwi

NIM: 135150201111200

DAFTAR RIWAYAT HIDUP

DATA PRIBADI

Nama	: Aprillia Arum Pratiwi
Tempat, Tanggal Lahir	: Malang, 23 April 1995
Alamat	: Dsn Lotekol RT 64 RW 07 Arjowilangun Kalipare Kab. Malang
Jenis Kelamin	: Perempuan
Agama	: Islam
Kewarganegaraan	: Warga Negara Indonesia
Status	: Belum Menikah
No Telp	: 0823-35849157

PENDIDIKAN FORMAL

2001-2007	SDN 03 Arjowilangun
2007-2010	SMP N 01 Kalipare
2010-2013	SMA ISLAM Kepanjen
2013-2018	Universitas Brawijaya

PENGALAMAN ORGANISASI

2014 Anggota Badan Internal Olahraga dan Seni (LSO Fakultas)

KATA PENGANTAR

Puji syukur peneliti panjatkan kepada Allah Subhanahu Wa Ta'ala atas limpahan rahmat dan pentunjuk-Nya, sehingga penulis dapat menyelesaikan skripsi yang berjudul "*ANALISIS PERBANDINGAN KINERJA ALGORITME DIJKSTRA, BELLMAN-FORD, DAN FLOYD-WARSHALL UNTUK PENENTUAN JALUR TERPENDEK PADA ARSITEKTUR JARINGAN SOFTWARE DEFINED NETWORK*".

Dalam penyusunan dan penelitian skripsi ini tidak lepas dari bantuan moral dan materiel dari berbagai pihak, maka peneliti mengucapkan terima kasih kepada:

1. Allah SWT atas cinta kasih-Nya kepada setiap hamba-Nya. Terutama kepada peneliti cinta-Nya yang tak pernah surut.
2. Kedua orang tua (Bapak dan Mamak) atas segala do'a, kasih sayang, nasihat, pengertian, perhatian, tiada hentinya bersabar dan senantiasa memberi semangat kepada peneliti hingga terselesaikannya skripsi ini.
3. Kepada kakak kandung saya mas Anang CS beserta istri (mbak Sherly Anjar Sinta) dan juga keponakan saya Fergha Bumandhala Byoma atas semangat, doa dan keceriaan yang diberikan penulis sehingga dapat kembali bersemangat ketika penulis dilanda kegalauan hingga penulis dapat menyelesaikan skripsi ini.
4. Kepada mbah Sues, Mbah Patemi, mbah Mulyani, Yoga, Mada, Bulek Sukati, Om Doni dan seluruh saudara pihak bapak dan mamak yang memberikan semangat dengan cara mereka sendiri dan memenuhi kebutuhan penulis sehingga penulis dapat menyelesaikan skripsi ini.
5. Bapak Wayan Firdaus Mahmudy, S.Si, M.T, Ph.D. selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya Malang.
6. Bapak Heru Nurwarsito, Ir., M.Kom. selaku Wakil Ketua I Bidang Akademik Fakultas Ilmu Komputer Universitas Brawijaya Malang.
7. Bapak Tri Astoto Kurniawan, S.T, M.T, Ph.D. selaku Ketua Jurusan Teknik Informatika Universitas Brawijaya Malang.
8. Bapak Sabriansyah Rizqika Akbar, S.T, M.Eng. selaku Ketua Program Studi Teknik Komputer Universitas Brawijaya Malang.
9. Bapak Widhi Yahya, S.Kom, M.Sc. selaku dosen pembimbing I yang telah memberikan pengarahan dan bimbingan kepada penulis, sehingga dapat menyelesaikan skripsi ini dengan baik.
10. Bapak Mahendra Data S.Kom, M.Kom. selaku dosen pembimbing II yang telah memberikan pengarahan dan bimbingan kepada penulis, sehingga dapat menyelesaikan skripsi ini dengan baik.

11. Segenap bapak dan ibu dosen yang telah mendidik dan mengarahkan ilmunya selama menempuh pendidikan di Fakultas Ilmu Komputer Universitas Brawijaya Malang.
12. Teman-teman Teknik Informatika angkatan 2013 yang telah banyak memberi bantuan dan dukungan selama penulis menempuh studi di FILKOM Universitas Brawijaya.
13. Teman-teman seperjuangan skripsi yang kami biasa menyebut 71/2 cm: Asika, Indi, Nuril, Helmi, Guntur, Ubaid, bambang, Okta. Dan juga teman-teman seperjuangan skripsi lainnya: Faris, Nuril Risqi, Indi RW dan Asika yang bersedia kosnya diinapi oleh penulis dan teman-teman seperjuangan lainnya.
14. Teman-teman kos wanita pak Tur (mbak Lenny, mbak Lusi, mbak Puput) yang memberikan semangat dan doa kepada penulis sehingga skripsi ini selesai.
15. Seluruh pihak yang tidak dapat diucapkan satu persatu, penulis mengucapkan banyak terima kasih atas segala bentuk dukungan dan do'a sehingga laporan skripsi ini dapat selesai.

Penulis menyadari bahwa dalam penyusunan skripsi ini masih terdapat banyak kekurangan, sehingga saran dan kritik yang membangun sanga penulis harapkan. Akhir kata penulis berharap skripsi ini dapat membawa manfaat bagi semua pihak yang membutuhkan.

Malang, Agustus 2018

Aprillia Arum Pratiwi

aprilliaarumpratiwi@gmail.com

ABSTRAK

Software defined network (SDN) merupakan konsep jaringan yang terpusat dan fleksibel dibandingkan dengan jaringan tradisional yang ada sekarang. SDN mulai dikembangkan beberapa tahun terakhir ini dan sudah banyak diimplementasikan salah satunya adalah routing jaringan. Routing merupakan proses pencarian jalur komunikasi yang digunakan untuk melewatkan paket yang dikirimkan pengirim ke penerima. Dalam pencarian jalur diperlukan algoritme routing yang akan menentukan rute terpendek. Pada penelitian ini algoritme routing yang digunakan untuk menentukan jalur terpendek yaitu algoritme *Dijkstra, Bellman-Ford*, dan *Floyd-warshall.* Ketiga algoritme tersebut akan diimplementasikan menggunakan emulator *Mininet* dan *controller Ryu*. Untuk penentuan jalur dibutuhkan penentuan bobot *link* atau *cost. Cost* yang diterapkan dalam jaringan SDN ditentukan dengan inputan manual. *Cost* pada penelitian ini berdasarkan dengan perhitungan bagi antara *reference bandwidth* sebesar 1000 Mbps dan *link bandwidth* yang menggunakan tiga jenis besaran kapasitas yaitu 10 Mbps, 100 Mbps, dan 1000 Mbps. Pengujian dilakukan dengan parameter yaitu validasi, *convergence time, throughput, recovery time* dan *packet loss*. Berdasarkan hasil validasi, sistem berjalan sesuai dengan perhitungan perhitungan manual masing-masing algoritme. Pada pengujian *convergence time*, *Dijkstra* lebih unggul dengan rata-rata 0,0087 detik dibandingkan *Bellman-ford* 0,0094 detik dan *floyd-warshall* 0,02025 detik. Pada pengujian *throughput* ketiga algoritme tidak terlalu memiliki perbedaan yang jauh. Berdasarkan pengujian *recovery time* algoritme *floyd-warshall* memiliki *recovery time* lebih cepat dari algoritme lain. Berdasarkan pengujian *packet loss Dijkstra* masih unggul dalam menangani *packet loss* saat pengiriman.

Kata kunci: *Software Defined Network, Ryu, Mininet, Algoritme Dikstra, Algorime Bellman-Ford, Algoritme Floyd-Warshall*

ABSTRACT

Software defined network (SDN) is a centralized and flexible network concept compared to traditional networks that exist today. SDN has been developed in the last few years and has been widely implemented, one of which is routing networks. Routing is the process of finding communication lines that are used to pass packets sent by the sender to the recipient. In line search, a routing algorithm is needed that will determine the shortest route. In this study the routing algorithm used to determine the shortest paths are Dijkstra, Bellman-Ford, and Floyd-warshall algorithms. The three algorithms will be implemented using the Mininet and Ryu controller emulators. For determining the path required the determination of the link or cost weight. The cost applied in the SDN network is determined by manual input. Cost in this study is based on the calculation of the reference bandwidth of 1000 Mbps and the bandwidth link that uses three types of capacity quantities, namely 10 Mbps, 100 Mbps, and 1000 Mbps. Testing is done with parameters, namely validation, convergence time, throughput, recovery time and packet loss. Based on the results of validation, the system runs according to the calculation of the manual calculation of each algorithm. In convergence time testing, Dijkstra was superior with an average of 0.0087 seconds compared to Bellman-ford 0.0094 seconds and Floyd-Warshall 0.02025 seconds. In testing the three throughput algorithms do not have far distinction. Based on recovery time testing, the Floyd-Warshall algorithm has faster recovery time than other algorithms. Based on the testing of packet loss Dijkstra is still superior in handling packet loss when sending.

Keywords: *Software Defined Network, Ryu, Mininet, Dikstra Algoritthm, Bellman-Ford Algoritthm, Floyd-Warshall Algoritthm*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar belakang

Dewasa ini teknologi jaringan internet khusunya pada bidang infrastruktur jaringan komunikasi mengalami perkembangan yang semakin pesat. Perkembangan jaringan komunikasi yang menghubungkan berbagai perangkat komunikasi dirancang untuk memenuhi kebutuhan akan komunikasi yang cepat dan efisien. *Internet* maupun jaringan komputer terbentuk dari berbagai perangkat yang biasa disebut dengan node yang dihubungkan oleh *edge*. Perangkat tersebut di dalamnya terdapat suatu node harus mencari jalur komunikasi yang tepat dari sejumlah jalur yang ada. Proses pencarian jalur ini disebut dengan *routing* (Ericko, 2016). Konfigurasi *routing* pada jaringan konvensional masih dilakukan secara individual, hal tersebut menyebabkan tidak fleksibel terhadap perubahan. Pada jaringan *Software Defined Network* (SDN) *routing* juga diperlukan dalam penentuan jalur komunikasi yang tepat untuk memaksimalkan kinerjanya. Jaringan *Software Defined Network* atau lebih dikenal dengan SDN merupakan salah satu evolusi teknologi jaringan yang sesuai dengan tuntutan yang sedang berkembang saat ini (Riza Abu S,2016).

Software Defined Network (SDN) memiliki konsep yang berbeda dengan jaringan tradisional. Pada jaringan tradisional *control plane* dan *data plane* digabung pada satu perangkat, sedangkan pada jaringan SDN *control plane* dipisahkan dari *data plane*. Pemisahan ini akan memudahkan dalam proses mengatur, mengelola dan melakukan monitoring jaringan (S.Sharma,et al.,2014). Konsep dari jaringan SDN telah menyederhanakan konsep jaringan tradisional yang ada sekarang, dimana kontrol jaringan pada sebuah *controller* yang membuat suatu jaringan mudah diatur dan lebih fleksibel. Hal tersebut dikarenakan pada jaringan SDN sebuah *controller* bersifat *programmable*, sehingga menjadikannya dapat diatur sesuai dengan kebutuhan (Kreutz,et al., 2015). *Controller* merupakan bagian dari SDN yang bertanggung jawab untuk mendefinisikan jaringan, mengatur masalah *availability*, laju *traffic* data, *routing* dan *forwarding* dan lain-lain. Dalam proses pencarian jalur komunikasi atau *routing* ini membutuhkan protokol *routing*. Suatu protokol *routing* memiliki tabel yang berisi informasi jalur yang akan dilalui oleh suatu paket data, dari informasi tersebut ditentukan suatu jalur yang akan digunakan untuk melewatkan paket berdasarkan alamat tujuannya. Dalam mengoptimalkan kinerja protokol *routing* diperlukan Algoritme *routing protocol* yang dapat menentukan rute terpendek pada berbagai macam topologi. Untuk membangun suatu *routing protocol* yang baik, maka diperlukan Algoritme *routing protocol* yang dapat menentukan jalur terpendek pada berbagai topologi tanpa melakukan konfigurasi ulang (Ulfa Kurniawati, 2016).

Penelitian sebelumnya tentang analisis perbandingan Algoritme *Dijkstra* dan *Bellman-Ford* pernah dilakukan oleh Ulfa Kurniawati dari Fakultas Ilmu Komputer, Universitas Brawijaya, pada 2016 lalu dalam penelitian yang berjudul "Analisis Perbandingan Algoritme *Dijkstra* dan *Bellman-Ford* dengan Menggunakan

Software Defined Network". Dalam penelitian tersebut penulis membandingkan kedua Algoritme dari hasil pengujian yang telah dilakukan pada topologi *Abilene*. Dari pengujian yang telah dilakukan menghasilkan Algortime *Dijkstra* mampu dieksekusi lebih cepat dari Algoritme *Bellman-Ford* (Ulfa Kurniawati, 2016). Penelitian selanjutnya membandingkan Algoritme *DJikstra* dan *Floyd-Warshall* yang dilakukan oleh Ibrahim mahasiswa Universitas Brawijaya. Pengujian tersebut dilakukan berdasarkan parameter uji yaitu *convergence time, throughput, recovery time, resource usage*. Hasil dari pengujian tersebut Algoritme *Djikstra* lebih unggul dari Algoritme *Floyd-Warshall* pada pengujian *convergence time*. Hasil pengujian *Link failure* pada *recovery time*, Algoritme *Floyd-Warshall* lebih unggul dari Algoritme *Djikstra* (Ibrahim, 2016).

Dari penelitian yang sudah ada, perlu dilakukannya perbandingan lebih lanjut dari Algoritme *Dijkstra*, *Bellman-Ford* dan *Floyd-Warshall* pada *Software Defined Network* (SDN) sebagai penentu jalur terpendek. Agar dapat mengetahui algoritme mana yang memiliki kinerja yang baik untuk diterapkan pada jaringan arsitektur *software defined network*, maka peneliti akan melakukan analisis pada algoritme *Dijkstra, Bellman-Ford* dan *Floyd-Warshall* sebagai perbandingan. Penerapan dari ketiga Algoritme tersebut dengan melihat parameter uji meliputi pengujian validasi, *convergence time, throughput, recovery time* dan *resource usage*. Ketiga algoritme akan dijalankan menggunakan *controller* RYU dan menggunakan emulator *Mininet*. Sehingga mendapatkan hasil pengujian yang dapat dibandingkan untuk menentukan algoritme manakah yang memiliki hasil paling optimal jika diterapkan pada jaringan *software defined network* (SDN).

1.2 Rumusan masalah

Adapun rumusan masalah yang akan dibahas pada penelitian ini adalah sebagai berikut:

1. Bagaimana kinerja Algoritme *Djikstra, Bellman-Ford* dan *Floyd-Warshall* pada arsitektur jaringan *Software Defined Network* berdasarkan parameter uji yaitu pengujian validasi, *convergence time, throughput, recovery time* dan *packet loss*?
2. Bagaimana analisis dan perbandingan dari penerapan ketiga Algoritme tersebut pada jaringan *Software Defined Network* ?

1.3 Tujuan

Adapun beberapa tujuan dari penelitian ini adalah sebagai berikut:

1. Melakukan pengujian dengan menerapkan Algoritme *Djikstra, Bellman-Ford* dan *Floyd-Warshall* berdasarkan pengujian validasi, *convergence time, throughput, recovery time* dan *packet loss*
2. Melakukan analisis dan membandingkan hasil kinerja dari penerapan ketiga Algoritme yang telah dilakukan dalam pengujian

1.4 Manfaat

Manfaat dari pelaksanaan penelitian ini adalah sebagai berikut:

1. Dapat menjadi alternatif model jaringan dan referensi pemilihan Algoritme *routing* untuk mengoptimalkan utilitas dari suatu jaringan lebih khususnya untuk jaringan *Software Defined Network*
2. Adapun manfaat bagi penulis ialah dapat mengasah keterampilan dalam bidang pemodelan jaringan komputer, lebih khususnya untuk jaringan *Software Defined Network*
3. Adapun manfaat bagi Fakultas Ilmu Komputer adanya penelitian ini dapat dijadikan materi pada pembelajaran pada mata kuliah Arsitektur Jaringan Terkini

1.5 Batasan masalah

Dalam penelitian ini diperlukan batasan masalah pada sistem yang akan dibuat. Hal ini bertujuan agar permasalahan yang dirumuskan dapat lebih terfokus. Adapun masalah pada laporan ini adalah sebagai berikut :

1. Merancang simulasi penerapan algoritme *Djikstra, Bellman-Ford* dan *Floyd-Warshall* pada *software defined network* (SDN)
2. Penerapan dari ketiga algoritme sebagai rekayasa pembentukan jalur paket pada jaringan *software defined network* (SDN)
3. *Controller* yang digunakan adalah *controller* RYU dengan bahasa pemrograman *Phyton*
4. Pengujian untuk mengukur performa dari Algoritme *Dijkstra, Bellman-Ford* dan *Floyd-Warshall* dengan menggunakan OpenFlow pada tiap topologinya
5. Penelitian dilakukan dengan menggunakan *emulator* Mininet
6. Melakukan pengujian dengan parameter uji yaitu pengujian validasi, *convergence time, throughput, recovery time* dan *packet loss*
7. Sistem operasi yang digunakan dalam penelitian adalah Linux Ubuntu 14.04 LTS

1.6 Sistematika pembahasan

Sistematika pembahasan dalam penulisan proposal skripsi ini dapat diuraikan sebagai berikut:

BAB I PENDAHULUAN

Menguraikan dan menjelaskan tentang latar belakang permasalahan, rumusan masalah, batasan-batasan masalah, tujuan, manfaat, serta sistematika pembahasan.

BAB II LANDASAN KEPUSTAKAAN

Bab landasan kepustakaan menguraikan kajian pustaka dan dasar teori yang mendasari masalah yang dalam penelitian.

BAB III METODOLOGI

Membahas tentang yang digunakan dalam penelitian yang terdiri dari studi *literature*, perancangan, implementasi, pengujian, analisis hasil, dan kesimpulan.

BAB IV PERANCANGA DAN IMPLEMENTASI

Pada bab ini akan dijelaskan tentang perancangan dan imlementasi sistem menentukan jalur terpendek dengan menggunakan Algoritme *Djikstra, Bellmand-Ford* dan *Floyd-Warshall* yang di terapkan dalam jaringan *Software Defined Network*. Menjelaskan cara instalasi perangkat lunak pendukung dalam pengerjaan penelitian ini, yaitu *Mininet*, *controller* Ryu, perancangan topologi dan Algoritme.

BAB V PENGUJIAN DAN ANALISIS

Bab pengujian dan analisis menjelaskan tentang pengujian dari implementasi menentukan jalur terpendek dengan menggunakan Algoritme *Djikstra, Bellmand-Ford* dan *Floyd-Warshall* yang di terapkan dalam jaringan *Software Defined Network* sesuai parameter uji yaitu pengujian validasi, *convergence time, throughput, recovery time* dan *packet loss*. Pada bab ini juga akan menjelaskan menganalisis hasil dari pengujian yang dilakukan.

BAB VI PENUTUP

Bab penutup berisikan tentang kesimpulan dari hasil pengujian berdasarkan implementasi sistem yang dilakukan. Dan juga memberikan saran untuk peneliti selanjutnya untuk menyempurnakan penelitian lebih lanjut lagi.

BAB 2 LANDASAN KEPUSTAKAAN

Pada bab ini peneliti akan membahas mengenai kajian pustaka dan teori-teori penting untuk menunjang dan menjadi acuan dalam penyusunan penelitian. Pada kajian pustaka akan berisi tentang perbandingan analisis yang pernah dilakukan sebelumnya. Dasar teori berisi penjelasan teori-teori yang diperlukan dalam penelitian yang dilakukan.

2.1 Kajian pustaka

Kajian pustaka berisikan mengenai perbandingan analisis yang pernah dilakukan dengan rencana penelitian yang akan dilakukan, berikut adalah perbandingannya:

Tabel 2. 1 Tabel Perbandingan penelitian

No	Nama Penulis, Tahun, dan Judul	Persamaan	Perbedaan	
			Penelitian terdahulu	Rencana penelitian
1.	Ibrahim Attamimi.(2017). Analisis Perbandingan Algoritme *Floyd-Warshall* dan *Dijkstra* untuk Menentukan Jalur Terpendek pada Jaringan Openflow. Fakultas Ilmu Komputer, Universitas Brawijaya	Membandingkan *Shortest path Problem Algorithm* pada jaringan *OpenFlow* menggunakan *controller* Ryu	Membandingkan Algortime *Floyd-Warshall* dan *Dijkstra*	Membandingkan Algoritme *Dijkstra, Bellman-Ford* dan *Floyd-Warshall*
2.	Ulfa Kurniawati. (2016). Analisis Perbandingan *Dijkstra* dan *Bellman-Ford* untuk Menentukan Jalur Terpendek dengan Menggunakan *Software Defined Network*. Fakultas Ilmu Komputer, Universitas Brawijaya	Membandingkan *Shortest path Problem Algorithm* pada jaringan *Software Defined Network*	Membandingkan Algoritme *Dijkstra* dan *Bellman-Ford* dengan menggunakan *controller* Pyretic	Membandingkan Algoritme Dijkstra, *Bellman-Ford* dan *Floyd-Warshall* dengan menggunakan *controller* Ryu
3.	Erico Lazuardi. (2016). Metode Pemilihan Jalur Routing Adaptif	Mengimplementasikan	Mengimplementasikan Algoritme *Dijkstra* dengan	Mengimplementasikan Algoritme

	Berdasarkan Kemacetan Jaringan dengan Algoritme *Dijkstra* pada *OpenFlow Network*	Algoritme *Dijkstra*	menggunakan *controller* Pyretic	*Dijkstra* menggunakan *controller* Ryu

Dari beberapa penelitian yang telah dijelaskan pada Tabel 2.1, penelitian ini fokus menggunakan tiga algoritme *routing Dijkstra, Bellman-Ford* dan *Floyd-Warshall.*

2.2 Dasar teori

Berdasarkan beberapa informasi yang didapat dari beberapa kajian pustaka, dalam "Analisis Perbandingan Algoritme *Djikstra, Bellman-Ford* dan *Floyd-Warshal* untuk Menentukan Jalur Terpendek pada Jaringan *Software Defined Network* (SDN)" terdapat beberapa dasar teori, antara lain :

2.2.1 *Software Defined Network* (SDN)

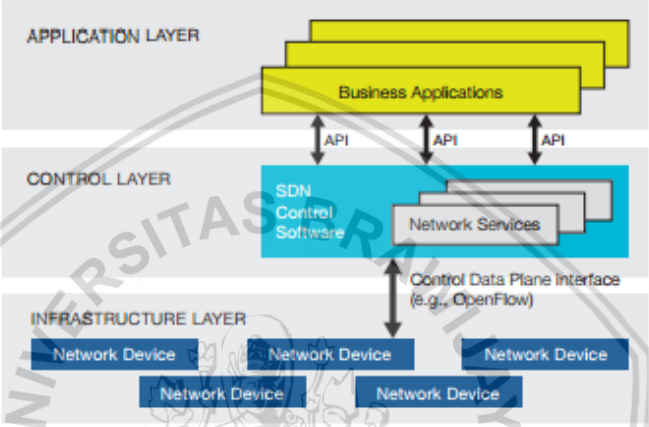
Tidak lama setelah *Sun Microsystems* merilis JAVA pada tahun 1995 dimulai muncullah teknologi *Software Defined Network*. Pada tahun 2008 teknologi *Software Defined Network* mulai dikembangkan di *UC Berkeley and Stanford University*. Kemudian *Open Networking Foundation* mempromosikan teknologi ini untuk memperkenalkan teknologi *Software Defined network* (SDN) dan *OpenFlow* pada tahun 2011.

Software Defined network (SDN) adalah istilah yang merujuk pada konsep baru dalam mendesain, mengelola dan mengimplementasikan jaringan untuk mendukung kebutuhan dan inovasi. Konsep dasar *Software Defined network* (SDN) adalah memisahkan antara *control* dan *forwarding plane,* serta kemudian melakukan abstraksi sistem dan meng-isolasi kompleksitas yang ada pada komponen atau sub-sistem dengan mendefinisikan antar-muka (*interface*) yang *standart*. Hampir sama seperti jaringan tradisional, jaringan SDN berkaitan erat dengan arsitektur perangkat *networking* seperti *router packet switch*, *local area network* (LAN) *switch* maupun perangkat *networking* yang lainnya. Pada dasarnya *control plane* merupakan bagian dari perangkat jaringan yang mengatur pemetaan jaringan, *routing table* dan lain sebagainya. Sedangkan *data plane* memiliki fungsi yang lainnya yaitu sebagai perangkat yang meneruskan paket-paket dari suatu port ke port yang lainnya dengan cara komunikasi yang telah ditentukan oleh *control plane* (McKeown, dkk, 2013)

Pemisahan tersebut diharapkan dapat memenej perangkat jaringan dengan melalui *controller*-nya saja. Untuk mewujudkan hal tersebut, dibutuhkan sebuah API yang digunakan untuk mengkoneksikan seluruh perangkat jaringan kedalam sebuah *controller* yang dapat diprogram sesuai dengan kebutuhan. Konfigurasi jaringan SDN dapat menciptakan jaringan dimana perangkat keras pengontrol lalu lintas data secara fisik dipisahkan dari perangkat keras yang bertugas sebagai *forwarding*.

Berikut merupakan keunggulan dari *Software Defined Network* :

1. Kinerja dari SDN memiliki kemampuan dalam memaksimalkan penggunaan perangkat jaringan
2. Peningkatan otomatisasi dan menejemen dengan menggunakan API
3. Berbagai inovasi yang menjadikan kemampuan jaringan semakin kompleks
4. Peluang baru untuk mendorong pendapatan dan diferensiasi oleh operator, perusahaan, vendor perangkat lunak independen untuk *programmability*
5. Dengan menerapkan kebijakan ditingkat sesi, perangkat, dan pengguna akan menjadikan kemapuan *control* jaringan menjadi lebih terinci



Gambar 2. 1 Arsitektur *Software Defined Network*

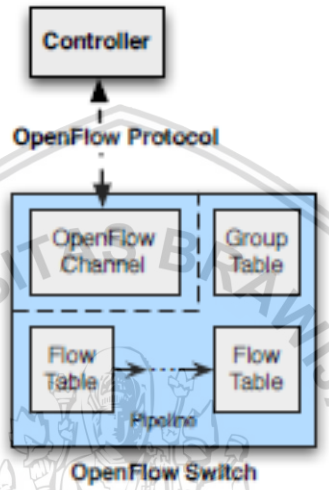
Gambar 2.1 menggambarkan pandangan logis dari arsitektur *Software Defined Network*. Infrastruktur *Software Defined Network* membentuk sebuah susunan yang terdiri dari tiga lapisan yaitu *Aplication Layer, Control layer,* dan *Infrastructure layer. Aplication layer* merupakan *layer* yang berisikan aplikasi-aplikasi seperti *mail server, web server*, maupun lainnya. *Control layer* memiliki peranan penting dalam mengendalikan sistem, ketika *controller* dipisahkan dari *data plane*. *Data plane* merupakan sekumpulan dari perangkat *networking* yang akan dikendalikan oleh *control plane*. *Infrastructure layer* dapat beroperasi sesuai perintah dari *controller*. Komunikasi antara perangkat dengan *controller* menggunakan *protocol* yang disebut dengan *OpenFlow*. (Ulfa K, 2016)

2.2.2 *OpenFlow*

Openflow merupakan sebuah *control interface* yang digunakan untuk menghubungkan antara lapisan *controller* dengan lapisan *forwarding* pada arsitektur *Software Defined Network*. Dengan menggunakan *protocol Openflow* komunikasi yang terjadi dilakukan oleh *controller* dengan perangkat-perangkat jaringan lainnya. Paket akan bergerak secara terpusat dengan menggunakan *Openflow*, maka jaringan dapat diprogram dengan cara independen dari *switch* individu dan *data center*.

Dari gambar 2.2 *Openflow switch* terdiri tiga bagian, yaitu yang pertama *flow table* yang berfungsi untuk memproses paket yang datang, pada *Openflow switch*

ada beberapa *flow table.* Kedua yaitu *secure channel* yang merupakan *interface* antara *controller* dan *Openflow swicth*, melalui *secure channel* ini terjadi pengiriman pesan untuk mengkonfigurasi *switch*. Dan yang ketiga yaitu *Openflow controller* yang merupakan standar untuk berkomunikasi dengan *Openflow switch. OpenFlow* merupakan *open standart* komunikasi protokol yang mampu melakukan pemisahan antara *control plane* dan *data plane* dari sebuah perangkat jaringan, serta mampu menciptakan komunikasi yang sangat baik antara *control plane* dan *data plane* (*OpenFlow Organization*, 2011)



Gambar 2. 2 OpenFlow

Sumber: Open Networking Spesification (2013)

Mekanisme kerja *protocol OpenFlow* saat *switch OpenFlow* menerima paket yang datang dan tidak memiliki kecocokan terhadap *flow table* yang ada maka *switch Openflow* akan meneruskan paket menuju *controller OpenFlow*. *Controller* memberikan respon terhadap paket tersebut berdasarkan *flow table*. Sehingga mekanisme komunikasi antara *control plane* dengan *data plane* yang diatur dengan melalui *controller* dapat memberikan respon yang sesuai terhadap setiap paket yang datang.

2.2.3 *Controller* SDN (RYU)

Software defined network memiliki beberapa *controller* yang digunakan untuk konfigurasi dalam komunikasi antara *aplication layer* dan *infrastruktur layer*. *Controller* berfungsi sebagai pusat kontrol atau logika jaringan. Semua kebijakan jaringan yang dilakukan seperti *switching, routing* maupun pengaturan jaringan lainnya dilakukan pada *controller*. Fungsional jaringan seperti *load balancing* dan *routing* dikembangkan melalui modul yang terdapat pada *controller* dengan bahasa masing-masing (Rizal Ahmad M, 2017). *Software defined network* memiliki beberapa *controller* yaitu *Floodlight, Open Daylight*, Trema,POX, dan Ryu.

Ryu merupakan perangkat lunak berbasis komponen yang didefinisikan pada topologi jaringan. Ryu menyediakan komponen perangkat lunak dengan API

terdefinisi dengan baik sehingga memudahkan pengembang untuk membuat aplikasi manejemen dan kontrol jaringan baru. Ryu mendukung berbagai protokol untuk mengelola perangkat jaringan seperti *OpenFlow, Netconf, OF-config* dan lain-lain. Pada *OpenFlow*, Ryu mendukung 1.0, 1.2, 1.3, 1.4, 1.5 dan *Nicira Extensions* sepenuhnya. Semua kodenya tersedia gratis di bawah lisensi *Apache* 2.0. (Ryu SDN *Framwork*)

2.2.3.1 *Network X*

NetworkX merupakan suatu paket perangkat lunak berbahasa *Python* yang dapat digunakan untuk membentuk, memanipulasi dan mempelajari struktur, dinamika serta fungsi dari jaringan hingga jaringan yang rumit/besar. Berbagai bentuk dari jaringan atau graf dapat dieksplorasi dan dianalisa dengan *NetworkX,* menggunakan suatu algortima yang dapat dimplementasikan untuk menghitung property atau sifat penting dari jaringan/graf tersebut (*NetworkX Developers, 2015)*.

Beberapa fitur-fitur yang disediakan oleh *NetworkX:*

- Mengikuti struktur data Bahasa *Python* untuk *graph* dan *multigraph.*
- Simpul (*node)* didalam *graph* bisa berupa "apa saja" dalam penelitian ini *node* adalah *Switch OpenFlow.*
- Algoritma *graph*
- *Multi-platform.*

2.2.4 *Mininet*

Mininet adalah suatu *software* emulator yang memungkinkan untuk melakukan *prototyping* pada jaringan yang luas dengan hanya menggunakan satu mesin. Dalam emulator mininet ini dilakukan perancangan topologi jaringan yang diinginkan. *Mininet* merupakan salah satu *software* yang paling sering digunakan pada SDN. *Meninet* menggunakan pendekatan virtualisasi untuk membuat suatu sistem. Seperti *host, Switch,* maupun *controller virtual*.

Mininet biasanya digunakan sebagai simulasi, verifikasi, *testing tool*, dan *resource*. Salah satu keunggulan adalah mampu membuat topologi sesuai yang diinginkan, selain itu juga mampu membuat topologi yang cukup kompleks, lebih besar, dan topologi *internet*. Fitur lain yang tidak kalah menarik dari *mininet* yaitu memungkinkan untuk kustomisasi paket *forwarding* sepenuhnya. Untuk mempermudah ketika menjalankan emulator *mininet*, dapat ditambahkan *miniedit* sebagai virtualisasinya. Dengan menggunakan *miniedit*, ketika merancang topologi tidak perlu melakukan *source code* antar *switch* di terminal. Dengan menggunakan *Miniediti* pengguna dimudahkan untuk membuat topologi pada *Mininet* dengan tanpa melakukan perintah yang ada pada terminal. Pada Miniedit hanya menggunakan sistem *drag and drop* untuk membuat topologi.

No	Ikon	Nama	Fungsi
1		Kursor	Memilih dan Memindahkan komponen seperti *switch* dan *controller*.

2		*Host*	Bertindak sebagai *host device* yang menggunakan layanan jaringan pada topologi.
3		*OpenFlow switch*	Bertindak sebagai *data plane* pada *SDN*.
4		Tradisional *Switch*	Bertindak sebagai *switch* tradisional yang meneruskan paket menggunakan pemetaan alamat MAC berdasarkan letak *port* bersambungnya.
5		Tradisional *Router*	Bertindak sebagai *router* tradisional yang mengambil keputusan penerusan paket berdasarkan protokol *routing*.
6		*Link*	Berfungsi untuk menghubungakan antara satu komponen dengan komponen yang lainnya.
7		*Controller*	Bertindak sebagai *Control Plane* pada *SDN*.

2.2.5 *Routing*

Routing adalah proses pada pemilihan jalur yang dilalui oleh paket data. Jalur terbaik sesuai dengan beban jaringan, panjang *datagram,* tipe layanan yang diminta dan pola lintasan. Terdapat beberapa jenis *routing* yaitu *minimal-routing*, *static-routing*, dan *dynamic routing*. *Minimal routing* adalah informasi minimum yang harus ada untuk *host* yang tersambung pada suatu *network*. *Minimal routing* terbentuk pada saat konfigurasi *interface.* Pada *static routing*, router meneruskan paket dari sebuah *network* ke *network* yang lainnya berdasarkan yang sudah ditentukan oleh *administrator*. Rute pada *static routing* tidak berubah kecuali jika diubah secara manual oleh *administrator*. Pada *dynamic routing*, router mempelajari sendiri rute terbaik yang akan ditempuh untuk meneruskan paket dari sebuah *network* ke *network* lainnya. *Administrator* tidak menentukan rute yang harus ditempuh oleh paket-paket tersebut. *Administrator* hanya menentukan bagaimana cara router mempelajari paket. Rute pada *dynamic routing* berubah sesuai dengan pelajaran yang didapatkan oleh *router*.

2.2.6 Algoritme *routing*

Algoritme *routing* merupakan mekanisme pencarian jalur yang efisien untuk pengiriman suatu paket. Sehingga paket di terima dengan lebih cepat oleh klien. Tugas utama dari algoritme *routing* yaitu mencari jalur terpendek dalam pengiriman suatu paket dengan waktu yang minimum.

Saat ini algoritme *routing* mulai berkembang dengan menawarkan mekanisme *routing* yang efisien. Salah satunya adalah algoritme *Djikstra* yang memiliki

mekanisme pencarian dengan menghitung seluruh cost yang berhubungan dengan *link* pada setiap jalur untuk mendapatkan *metric* jalur-jalur yang terhubung. Sedangkan *Bellman-Ford* memberikan *router-router* kemampuan untuk mempublikasikan semua jalur yang diketahui (router bersangkutan). Dan Algoritme *routing Floyd-warshall* melakukan pemecahan masalah dengan memandang solusi yang akan diperoleh sebagai suatu keputusan yang saling terkait, yang artinya pengambilan keputusan mengacu pada kesimpulan-kesimpulan sebelumnya.

2.2.6.1 Algoritme *Djikstra*

Algoritme *Dijkstra* ditemukan oleh Edser W. *Dijkstra* dan dipublikasikan pada tahun 1959 pada sebuah jurnal *Numerische Matematik*. Algoritme *Dijsktra* adalah sebuah algoritme *greedy* yang dipakai dalam memecahkan permasalahan jarak terpendek pada sebuah graf dengan bobot sisi (*edge weights*) yang bernilai tak-negatif (*morris*). Ada beberapa kasus pencarian lintasan terpendek yang diselesaikan menggunakan Algoritme *Dijkstra*, yaitu: pencarian lintasan terpendek antara dua buah simpul tertentu (*a pair shortest path*), pencarian lintasan terpendek antara semua pasangan simpul (*all pairs shortest path*), pencarian lintasan terpendek dari simpul tertentu ke semua simpul yang lain (*single-source shortest path*), serta pencarian lintasan terpendek antara dua buah simpul yang melalui beberapa simpul tertentu (*intermediate shortest path*).

Penggunaan strategi *greedy* pada Algoritme *Dijkstra* adalah: Pada setiap langkah, ambil sisi berbobot minimum yang menghubungkan sebuah simpul yang sudah terpilih dengan sebuah simpul lain yang belum terpilih. Lintasan dari simpul asal ke simpul yang baru haruslah merupakan lintasan yang terpendek di antara semua lintasannya ke simpul-simpul yang belum terpilih. Berikut merupakan pseudocode algoritme Djikstra secara umum:

```
1.  function Dijkstra(Graph, source):
2.      for each vertex v in Graph:
3.      dist[v] := infinity ;
4.        previous[v] := undefined ;
5.      end for
6.      dist[source] := 0 ;
7.      Q := the set of all nodes in Graph
8.      while Q is not empty:
9.        u := vertex in Q with smallest distance in dist[] ;
10.       remove u from Q ;
11.       if dist[u] = infinity:
12.         break ;
13.       end if
14.       for each neighbor v of u:
15.         alt := dist[u] + dist_between(u, v) ;
16.         if alt < dist[v]:
17.           dist[v] := alt ;
18.           previous[v] := u ;
19.           decrease-key v in Q;
20.         end if
21.       end for
22.     end while
23.   return dist;
```

Gambar 2. 3 *Pseudocode Algoritme Djikstra*

Sumber: (Kurose & Ross, 2012)

Berdasarkan gambar 2.3 dapat dijelaskan bahwa, jika inputannya tujuan adalah sumber maka set nilai 0 dan jika tujuan bukan sumber maka set nilai tak terhigga. Kemudian inputkan tujuan untuk dimasukkan ke dalam antrian (*queue*). Mencari jalur terpendek antara 2 node u (sumber) dan v (tujuan). Setiap nilai v yang dekat dengan u, maka pencarian dilakukan dengan membandingkan nilai debelumnya. Jika jarak v stsu dist[v] lebih besar dari dist[u], maka dalam tabel *queue* dirubah dengan jalur yang ter pendek adalah dist[u] begitu seterusnya sehingga mencapai semua node.

2.2.6.2 Algoritme *Bellman-Ford*

Bellman-Ford menghitung jarak terpendek (dari satu sumber) pada sebuah graf berbobot. Maksud dari satu sumber adalah menghitung semua jarak terpendek yang berawal dari satu titik node. Algoritme *Bellman-Ford* hanya digunakan jika ada sisi (*edge*) yang berbobot negatif.

Dalam routing algoritme ini digunakan dalam *distance vector routing protocol*, misalnya *Routing Information Protocol* (RIP). Algoritme *Bellman-Ford* didistribusikan karena melibatkan jumlah node (*router*) dalam *Autonomous system*, koleksi jaringan IP biasanya dimiliki oleh ISP.

Langkah-langkah dari Algoritme *Bellman-Ford* adalah sebagai berikut :

1. Setiap node menghitung jarak antara dirinya dan semua node lain dalam AS, dan menyimpan informasi sebagai sebuah table.
2. Setiap node mengirimkan table ke semua node tetangga.
3. Ketika sebuah node menerima table jarak dari tetangganta, ia menghitung rute terpendek ke semua node lainnya dan update table sendiri untuk menggambarkan perubahan yang terjadi.

```
1   Function bellmanFord(G,S)
2      For each vertex V in G
3               Distance[V]<-infinite
4               Previous[v]<-NULL
5       Distance[S]<-0
6
7      For each vertex V in G
8            For each edge(U,V)in G
9               tempDistance<-distance[u]+edge_weight(U,V)
10              if tempDistance<distance[V]
11              distance[V]<-tempDistance
12              previous[V]<- U
13
14     for each edge (U,V) in G
15          if distance[U] + edge_weight(U,V)<distance[V]
16
17  return distance[],prevous[]
```

Gambar 2. 4 *Pseudocode Algoritme Bellman-Ford*

2.2.6.3 Algoritme *Floyd-Warshall*

Algoritme *Floyd-Warshall* adalah salah satu varian dari pemrograman dinamis, yaitu suatu metode yang melakukan pemecahan masalah dengan memandang solusi yang akan diperoleh sebagai suatu keputusan yang saling terkait. Artinya solusi-solusi tersebut dibentuk dari solusi yang berasal dari tahap sebelumnya dan ada kemungkinan solusi lebih dari satu. Hal yang membedakan pencarian solusi menggunakan pemrograman dinamis dengan Algoritme *greedy* adalah bahwa keputusan yang diambil pada tiap tahap pada Algoritme *greedy* hanya berdasarkan pada informasi yang terbatas sehingga nilai optimum yang diperoleh pada saat itu. Jadi pada Algoritme *greedy*, kita tidak memikirkan konsekuensi yang akan terjadi seandainya kita memilih suatu keputusan pada suatu tahap.

Berikut merupakan *pseudocode* algoritme *Floyd-Warshall* secara umum :

```
1  Function fw(int[1...n.1..n] graph)
2  {
3  //inisialisasi
4  var int[1...n.1..n]jarak := graph
5  var int[1...n.1..n]sebelum
6  for i from 1 to n
7  for j from 1 to n
8  if jarak[I,j]<t~
9        sebelum[I,j] := i
10
11 //perulangan
12 for k from 1 to n
13 for j from 1 to n
14 for i from 1 to n
15 if jarak[i,j] > jarak[i,k]+jarak[k,j]
16    jarak[i,j]=jarak[i,k]+jarak[k,j]sebekum[I,j]=sebelum[
17       k,j]
18 return jarak
19 }
```

Gambar 2. 5 *Pseudocode* Algoritme *Floyd-Warshall*

Pada iterasi ke-1, setiap sel matriks dilakukan pengecekan apakah jarak antar dua titik mula mula lebih besar dari penjumlahan antar jarak titik asal ke titik tujuan (titik tujuan=iterasi ke-1) dengan jarak titik asal (titik asal=iterasi ke-1) ke titik tujuan. Dengan kata lain apakah jarak [i,j] > jarak [i,k] + jarak [k,j]. Jika iya maka jarak antar dua titik mula mula diganti dengan penjumlahan antar jarak titik asal ke titik tujuan (titik tujuan=iterasi ke-1) dengan jarak titik asal (titik asal=iterasi ke-1) ke titik tujuan (jarak [i,k] + jarak [k,j]). Jika tidak, maka jarak yang digunakan yaitu jarak antar dua titik mula mula (jarak [i,j]). Proses iterasi dilakukan hingga pada iterasi terakhir (jumlah iterasi=jumlah total titik).

2.2.7 Parameter kinerja jaringan

Kriteria yang penting dari sudut pandang pemakaian jaringan adalah keandalan. Keandalan yang dimaksud yaitu kriteria pengukuran seberapa mudah suatu sistem mengalami suatu gangguan, terjadi kegagalan atau sistem beropeprasi dengan tidak semestinya. Keandalan ialah ukuran statisitik kualitas komponen dengan menggunakan strategi pemeliharaan, kuantitas redudansi,

perluasan secara geometris dan kecenderungan statis dalam merasakan sesuatu secara tidak langsung tentang bagaimana suatu paket ditransmisikan oleh sistem tersebut. Dalam mengukur kinerja suatu jaringan dengan menentukan beberapa pengujian berdasarkan parameter pengujiannya. Beberapa parameter uji yang dapat dilakukan untuk mengukur kinerja jaringan ialah berdasarkan *convergence time, throughput, recovery time* dan *packet loss.*

- **Pengujian Validasi**

Pada pengujian validasi ini dilakukan dengan menghitung secara matematis dari node sumber ke node tujuan. Pengujian ini dilakukan untuk mengetahui apakah program yang dibuat sesuai dengan hitungan manual matematis pada masing-masing algortima.

- ***Convergence time***

Convergence time merupakan waktu yang dibutuhkan sebuah jaringan untuk mencapai keadaan stedy state pada semua *link*. Perhitungan *convergence time* dilihat dari ketika *switch* menerima paket dan meminta *table routing* ke *controller* sampai controller memberikan *table routing* kepada *switch*.

- ***Throughput***

Throughput adalah kecepatan data sebenarnya yang terukur pada suatu waktu tertentu. Pada penelitian ini perhitungan *throughput* dilihat dari seberapa besar kecepatan data yang di diperoleh untuk mengirimkan paket dari suatu *host* ke *host* lain.

- ***Recovery Time***

Recovery time adalah waktu yang dibutuhkan *controller* untuk menemukan jalur baru ketika terjadi *link failure* pada saat pengiriman paket.

- ***Packet Loss***

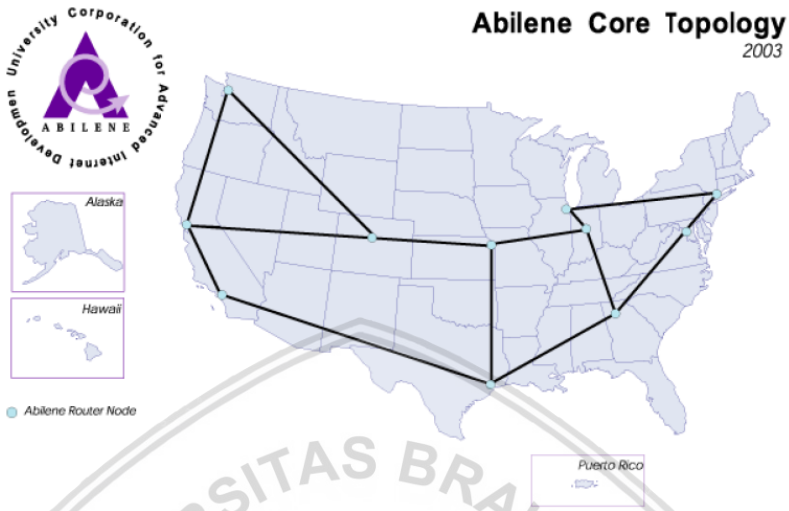
Packet loss terjadi ketika satu atau beberapa paket data yang melintasi suatu jaringan komputer gagal mencapai tujuannya (Wikipedia, 2017). Packet loss biasanya terjadi disebabkan oleh *traffic* yang padat pada jaringan.

2.2.8 Topologi *Abilene*

Topologi *Abilene* merupakan suatu topologi *backbone* dengan kinerja tinggi yang di ciptakan oleh *Internet Community* dengan *Network Operations Center* (NOC) di Universitas Indiana pada akhir tahun 1990. Namun saat ini topologi *Abilene* sudah tidak digunakan lagi, karena pada tahun 2007 telah ditingkatkan menjadi *Internet2 Network*.

Gambar 2.6 merupakan peta topologi *Abilene.Abilene* di ambil dari salah satu nama stasiun yang ada di kota Kansas, karena topologi *Abilene* adalah implementasi dari rute kereta api yang terdapat di Amerika. Tujuan di buatnya topologi *Abilene* yaitu untuk mencapai 10 Gbps konektivitas antar *node* dan 100 Mbps konektivitas antar *host* dan *node*. *Abilene* merupakan *private network* yang

diperuntukkan bagi penelitian serta pendidikan, namun biasanya tidak hanya itu saja, karena akses alternatif akan diberikan ke banyak sumber daya melalui *public network*.



Gambar 2. 6 Topologi *Abilene*

Sumber: (Stanford, University IT, 2015)

BAB 3 METODOLOGI

Pada bab ini peneliti akan menjelaskan tentang metode yang digunakan dan langkah-langkah penyusunan perancangan sistem yang akan dilakukan dalam penelitian ini.

3.1 Metode Penelitian

Bagian ini menjelaskan langkah-langkah yang akan dilakukan dalam penyusunan tugas akhir ini. Beberapa tahapan sebagai metodologi penelitian yaitu perumusan masalah, studi literatur, perancangan, implementasi dan pengujian, analisis dan hasil, dan kesimpulan. Berikut diagram alir tahapan metodologi penelitian yang digunakan:

Gambar 3. 1 Diagram Alir Metode Penelitian

Berdasarkan gambar 3.1 dapat di uraikan langkah-langkah metodologi penelitian yang akan dilakukan, yaitu:

1. Perumusan masalah dalam penelitian ini dirumuskan berdasarkan masalah yang sedang terjadi. Dari perumusan masalah yang disusun maka, peneliti

dapat mengambil langkah untuk memberikan slusi dari permasalahan yang teah dirumuskan.
2. Studi literatur penelitian sebelumnya yang terkait, *Software Defined Network, OpenFlow, controller* (RYU), *mininet, Routing*, Algoritme *Routing*, Parameter kinerja jaringan, dan topologi
3. Perancangan atau desain sistem, topologi, dan lingkungan untuk menerapkan algoritme routing pada SDN
4. Implementasi sistem pada *OpenFlow S*DN
5. Pengujian sistem routing pada jaringan SDN
6. Analisis hasil pengujian berdasarkan parameter uji yang telah ditentukan
7. Penarikan kesimpulan bersadarkan hasil analisis pengujian yang dilakukan terhadap sistem.

3.2 Studi literatur

Pada penelitian ini, dilakukan studi literatur untuk menjadi dasar dan landasan terhadap perancangan dan implementasi yang digunakan untuk menunjang penelitian. Berbagai studi literature yang dilakukan terhadap dasar-dasar SDN dan *OpenFlow* beserta aplikasi dan pendukung penerapannya. Dasar teori tersebut diperoleh dari buku, *e-book*, dan dokumentasi projek. Berikut dasar teori yang digunakan :

1. *Software Defined Network* (SDN)
2. *Mininet*
3. *OpenFlow*
4. *Controller* SDN (RYU)
5. *NetworkX*
6. Routing
7. Algoritme *Djikstra*
8. Algoritme *Bellman-Ford*
9. Algoritme *Floyd-Warshall*

3.3 Analisis kebutuhan

Analisis kebutuhan bertujuan untuk memperoleh semua kebutuhan yang diperlukan dalam penelitian. Analisis kebutuhan pada penelitian ini dibagi menjadi kebutuhan fungsional dan kebutuhan non-fungsional.

3.3.1 Kebutuhan fungsional

Kebutuhan fungsional berisi tentang gambaran kemampuan dari sistem. Kebutuhan fungsional dalam penelitian ini diantaranya:

1. Sistem dapat menentukan rute terpendek dari sejumlah *switch* yang disediakan.
2. Ketiga Algoritme yang digunakan dapat menghasilkan nilai yang dapat diukur sebagai parameter perbandingan

3.3.2 Kebutuhan non-Fungsional

Kebutuhan non-fungsional ini akan menjelaskan tentang kebutuhan yang berhubungan dengan perangkat keras maupun perangkat lunak agar dapat mendukung pembuatan sistem dan melancarkan penelitian ini. Berikut adalah kebutuhan non-fungsional dari sistem, yaitu:

a. Perangkat keras :

   Sebuah PC, dengan spesifikasi:

   - CPU = Intel core i3-350, 2,26 GHz
   - RAM = 3GB
   - Harddisk =320 GB

b. Perangkat lunak :

   - Linux Ubuntu 14.04 32-bit : sebagai sistem operasi yang digunakan oleh system
   - *RYU* digunakan sebagai *controller*. Bahasa pemrograman yang digunakan adalah *Python*
   - Mininet : sebagai *emulator* yang digunakan untuk membangun topologi
   - *OpenFlow* : sebagai *protocol routing*
   - *Iperf, ping, bwm-ng :* sebagai *tools* aplikasi untuk melakukan pengujian pada sistem

3.4 Perancangan simulasi

Perancangan simulasi ini menggambarkan mekanisme perancangan sistem simulasi yang akan dibangun dalam penelitian. Pada sub bab perancangan ini juga akan menjelaskan mengenai langkah-langkah perancangan suatu sistem yang akan dapat memenuhi kebutuhan berdasarkan analisis kebutuhan yang telah dilakukan sebelumnya. Sistem yang dirancang diharapkan mampu mengatasi kelemahan algoritme *routing* yang pernah diimplementasikan sebelumnya.

Dalam pengerjaan penelitian ini dimulai dengan perancangan sistem yang akan dibangun. Sistem yang akan dibangun dan diujikan merupakan sistem simulasi. Penelitian ini menggunakan Algoritme *Djikstra, Bellman-Ford* dan *Floyd-Warshall* yang akan diterapkan dan diujikan pada jaringan *Software Defined Network*. Topologi jaringan yang akan diimplementasikan pada program emulator jaringan *Mininet*. Kemudian dilakukan instalasi aplikasi pendukung sistem yaitu *Mininet* dan *controller* Ryu. Pengecekkan keberhasilan instalasi *Mininet* dengan perintah “Sudo mn”, sedangkan untuk *Controller* Ryu dengan perintah “Ryu-Manager”. Pengecekkan konektivitas dilakukan dengan perintah “pingall” pada Mininet. Tahap selanjutnya pengujian sistem sesuai dengan skenario. Pengujian dilakukan menurut parameter uji yaitu pengujian validasi, *convergence time, throughput, recovery time* dan *packet loss.* Setelah dilakukannya pengujian, maka dilakukan analisis perbandingan pada setiap algoritme dan pengambilan data hasil.

3.5 Perancangan Algoritme

Perancangan algoritme akan membahas mengenai proses perancangan algoritme routing pada arsitektur jaringan *software defined network*. Penelitian ini menerapkan algoritme *Dijkstra, Bellman-Ford* dan *Floyd-Warshall*.

3.5.1 Perancangan Algoritme *Djikstra*

Perancangan algoritme *Dijkstra* akan menjelaskan bagaimana mekanisme pencarian jalur pada algoritme *Dijkstra*. Algoritme *Dijkstra* melakukan pencarian jalur dengan melakukan perhitungan cost pada setiap jalur sebelum menentukan jalur terpendek yang akan ditentukan.

3.5.2 Perancangan Algoritme *Bellman-Ford*

Perancangan Algoritme *Bellman-Ford* akan mejelaskan mekanisme pencarian rute pada Algoritme *Bellman-Ford.* Pencarian jalur algoritme *Bellman-Ford* dengan menanyakan nilai *cost* tetangga, kemudian saling bertukar informasi *cost*. Selanjutnya melakukan perhitungan *cost* pada setiap jalurnya sebelum menentukan jalur terpendek. Ketika terjadi update algoritme *Bellman-Ford* akan melakukan saling memberi informasi antar *switch.*

3.5.3 Perancangan Algoritme *Floyd-Warshall*

Perancangan Algoritme *Floyd-Warshall* akan mejelaskan mekanisme pencarian rute pada Algoritme *Floyd-Warshall*. Algoritme *Floyd-Warshall* melakukan perhitungan nilai cost terkecil pada seluruh jalur yang menghubungkan sebuah pasangan titik dan perhitungan tersebut dilakukan sekaligus pada semua pasangan titik. Algoritme *Floyd-Warshall* memilih jalur yang optimum untuk dilalui pertama kali.

3.5.4 Perancangan topologi

Pada perancangan topologi ini membahas tentang topologi yang akan digunakan dalam melakukan implementasi. Perancangan topologi ini dibuat menggunakan emulator *mininet*. Untuk memvisualisasikan topologi yang akan dibangun dapat dibantu pembuatannya dengan aplikasi *Miniedit*. *Miniedit* merupakan aplikasi visualisasi yang sering digunakan pada *Mininet* untuk mendemonstrasikan sistem simulasi yang dibangun.

Topologi yang akan dibangun merupakan topologi jaringan yang diterapkan pada jaringan di dunia nyata. Topologi *Abilene* merupakan topologi yang menyerupai wilayah pada negara Amerika Serikat. Pada topologi jaringan *Abilene* akan terdapat 1 controller dengan 11 *switch* yang saling terhubung dan masing-masing *switch* memiliki 1 host. Dengan topologi yang akan dibangun tersebut diharapkan mampu memberikan gambaran umum pada sistem yang akan dibangun. Dalam perancangannya akan ditentukan host yang satu bertindak sebagai *source* selaku pengirim paket data dan host satu yang lain akan bertindak sebagai *destination* selaku penerima paket data.

3.6 Implementasi

Implementasi sistem dilakukan berdasarkan perancangan yang telah dibuat. Implementasi dalam penelitian ini meliputi:

1. Melakukan instalasi simulator mininet dalam sistem operasi Linux
2. Melakukan instalasi RYU
3. Membuat program mekanisme *routing* jaringan dengan menggunakan Algoritme *Dijkstra, Bellman-Ford* dan *Floyd-Warshall*
4. Membangun topologi jaringan dan melakukan routing pada algoritme *Dijkstra, Bellman-Ford* dan *Floyd-Warshall*
5. Mengetahui hasil analisis perbandingan ketiga algoritme berdasarkan parameter uji pengujian validasi, *convergence time, throughput, recovery time* dan *packet loss*

3.7 Pengujian

Rencana pengujian dilakukan dengan cara mensimulasikan ketiga algoritme *routing* diantaranya adalah Algoritme *Dijksra, Bellman-Ford,* dan *Floyd-Warshall* yang telah diimplementasikan pada jaringan *Software Dedined Network* (SDN). Pengujian dalam penelitian ini fokus pada analisis kebutuhan fungsional. Pengujian fungsional berfungsi untuk melihat apakah kebutuhan fungsional sistem berjalan sesuai dengan perancangan atau tidak. Pengujian yang dilakukan adalah melakukan penilaian dan evaluasi mekanisme sebelum dan sesudah penerapan sistem. Pengujian ini membandingkan algoritme *Dijkstra, Bellamn-Ford* dan *Floyd-Warshall* dengan menggunakan *Software Defiend Nework*. Topologi yang digunakan menggunakan 1 (satu) *controller* dengan 11 (sebelas) *switch* yang saling terhubung dan masing-masing switch memiliki 1 (satu) host. Pengujian diawali dengan menentukan node sumber (*source*) dan node tujuan (*destination*). Pengujian dilakukan dengan mengamati pemilihan jalur pada setiap hasil dari dijalankannya ketiga Algoritme dan juga menganalisis hasil nilai parameter uji pengujian validasi, *convergence time, throughput, recovery time* dan *packet loss.*

- **Pengujian Validasi**

Pada pengujian validasi ini dilakukan dengan menghitung secara matematis dari node sumber ke node tujuan. Pengujian ini dilakukan untuk mengetahui apakah program yang dibuat sesuai dengan hitungan manual matematis pada masing-masing algortime.

- ***Convergency time***

Pada penelitian ini pengujian dilakukan dengan parameter *convergence time* yang merupakan waktu yang dibutuhkan *controller* untuk membentuk sebuah tabel *routing*. Perhitungan *convergence time* dilihat dari ketika *switch* menerima paket dan meminta tabel *routing* ke *controller* sampai *controller* memberikan table routing kepada *switch*. Pengujian dilakukan dengan melakukan pengiriman paket ICMP (*Internet Control Message Protocol*) menggunakan *command "*ping" dari host 9 yang berperan sebagai host sumber ke host 1 sebagai host tujuan.

Kemudian setelah *controller* berhasil membentuk jalur akan muncul waktu dengan satuan detik yang merupakan *convergence time.* Pengujian dilakukan dengan varian jumlah *switch* yang dikategorikan sedikit, sedang, dan banyak.

- ***Throughput***

Throughput adalah kecepatan data sebenarnya yang terukur pada suatu waktu tertentu. Pada penelitian ini perhitungan *throughput* dilihat dari seberapa besar kecepatan data yang di diperoleh untuk mengirimkan paket dari suatu *host* ke *host* lain dalam melakukan pengiriman paket TCP. Pengiriman paket TCP dilakukan dengan *command* "*iperf*" dari host 9 sebagai *client* dan host 1 sebagai *server*. Pengujian dilakukan dengan varian jumlah koneksi *client* yang dikategorikan sedikit, sedang dan banyak.

- ***Recovery Time***

Recovery time adalah waktu yang dibutuhkan controller untuk menemukan jalur baru ketika terjadi link failure pada saat pengiriman paket. Pada pengujian ini akan dilakukan pengiriman paket dari *client* ke *server* kemudian akan dilakukan pemutusan jalur atau *link failure.* Waktu yang dibutuhkan untuk menemukan jalur baru itulah yang akan menjadi *recovery time*. . Pengujian dilakukan dengan melakukan pengiriman paket ICMP (*Internet Control Message Protocol*) menggunakan *command "*ping" dari host 9 yang berperan sebagai host sumber ke host 1 sebagai host tujuan. Kemudian setelah *controller* berhasil membentuk jalur, akan dilakukan *link down* pada salah satu *link*. Kemudian *controller* membentuk jalur terbaru akan muncul waktu dengan satuan detik yang merupakan *recovery time.* Pengujian dilakukan dengan varian jumlah *switch* yang dikategorikan sedikit, sedang, dan banyak.

- ***Packet Loss***

Packet loss terjadi ketika satu atau beberapa paket data yang melintasi suatu jaringan komputer gagal mencapai tujuannya (Wikipedia, 2017). Packet loss biasanya terjadi disebabkan oleh *traffic* yang padat pada jaringan. Pada pengujian ini kan dilakukan dengan mengiriman paket UDP dari client ke server. Pengiriman paket UDP dilakukan dengan *command* "*iperf*" dari host 9 sebagai *client* dan host 1 sebagai *server*. Pengujian dilakukan dengan varian jumlah koneksi *client* yang dikategorikan sedikit, sedang dan banyak.

Analisis hasil dilakukan setelah memperoleh data dari pengujian. Analisis hasil dilakukan untuk mengukur hasil dari kinerja sistem. Hasil dari analisis akan berupa pengujian validasi, *convergence time, throughput, recovery time* dan *packet loss* dengan menggunakan Algoritme *Dijkstra, Bellman-Ford* dan *Floyd-Warshall*. Dengan memfokuskan penelitian pada perbandingan performa dari ketiga Algoritme tersebut pada jaringan *Software Defined Network*.

3.8 Analisis Hasil

Pada penelitian ini analisi hasil dilakukan ketika setelah mendapat data dari hasil pengujian pada algoritme *Dijkstra, Bellman-Ford* dan *Floyd-Warshall.* Analisis

dilakukan untuk mengukur kinerja sistem yang diterapkan apakah sistem tersbut sudah berjalan sesuai dengan tujuan penelitian.

3.9 Kesimpulan

Pada pengambilan kesimpulan akan dilakukan setelah seluruh tahapan perancangan, implementasi dan pengujian telah selesai dilakukan. Pengambilan kesimpulan akan diambil dari hasil analisis pengujian yang telah dilakukan. Tahap kesimpulan juga ditambahkan saran untuk memberikan masukan sebagai bahan pertimbangan untuk pengembangan pada penelitian selanjutnya.

BAB 4 PERANCANGAN DAN IMPLEMENTASI

Pada bab ini peneliti menjelaskan tentang perancangan sistem routing yang dilakukan dalam simulasi routing pada jaringan *Software Defined Network*. Perancangan yang dilakukan berpedoman pada metodologi yang telah dibahas sebelumya.

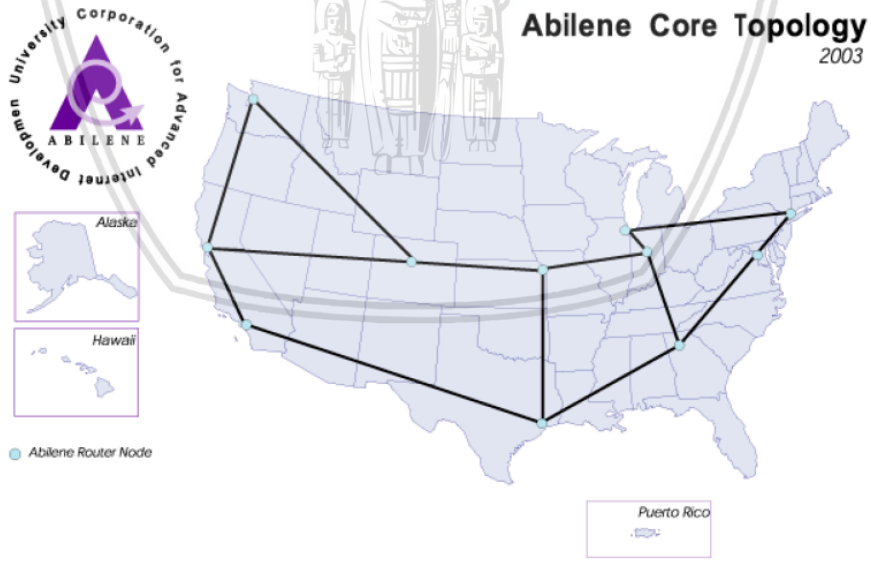
4.1 Perancangan

Perancangan dilakukan untuk merancang sistem yang dibangun berdasarkan kebutuhan yang diperlukan untuk menunjang berjalannya sistem.

4.1.1 Perancangan Topologi

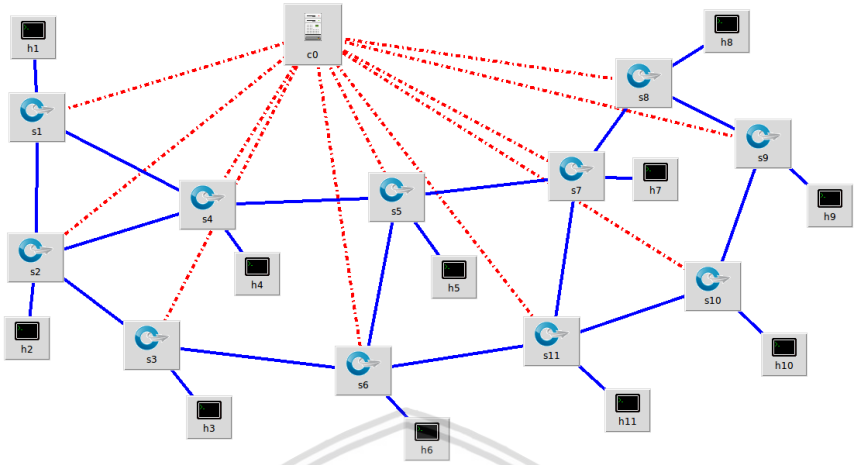
Topologi digunakan sebagai visualisasi jaringan. Pada proses *routing*, topologi dirancang dengan memiliki jalur yang bersifat redundan yang dapat dilalui suatu node. Jalur yang bersifat redundan apabila dalam suatu jaringan memiliki node sumber dan node tujuan. Dapat dikatakan redundan juga ketika terdapat beberapa jalur yang dapat dilalui oleh node sumber dan node tujuannya.

Untuk pengujian pencarian jalur dilakukan dengan menggunakan skenario topologi dimana *routing* diterapkan pada jaringan di dunia maya. Topologi *Abilene* merupakan topologi yang telah digunakan pada Negara Amerika Serikat dan memiliki kemiripan dengan wilayahnya. Peta jaringan *Abilene* dapat dilihat pada gambar 4.1 dan 4.2 merupakan pemetaan yang didesain pada *Mininet*.



Gambar 4. 1 Peta Jaringan *Abilene*

Sumber: (Stanford, University IT, 2015)



Gambar 4. 2 Design Topologi Jaringan *Abilene* pada *Mininet*

Pada topologi *Abilene* yang sudah didesain pada *mininet* terdapat 11 (sebelas) *switch* dan pada setiap *switch* memiliki 1 (host).

4.2 Perancangan Simulasi

Berikut merupakan langkah perancangan suatu sistem yang akan dapat memenuhi kebutuhan berdasarkan analisis kebutuhan yang telah dilakukan pada bab 3. Sistem yang dikembangkan merupakan sistem simulasi dari suatu jaringan *Software Defined Network* yang akan dianalisis dengan menguji algortima *routing* yang diterapkan. Berikut merupakan tahapan yang perlu dilakukan untuk dapat mendukung berjalannya sistem simulasi sesuai dengan yang diharapkan.

4.2.1 Instalasi

Instalasi memuat langkah yang perlu dilakukan untuk memenuhi kebutuhan yang diperlukan dalam pembangunan sistem *routing*. Beberapa perangkat lunak yang menjadi pendukung dalam pembangunan sistem *routing* beserta cara instalasinya adalah sebagai berikut:

4.2.1.1 *Mininet*

Mininet merupakan simulasi jaringan yang digunakan untuk mengembangkan sistem dengan pendekatan virtualisasi. Berikut merupakan langkah instalasi *mininet* pada *Operating Sistem Ubuntu*.

1. Langkah pertama yang dilakukan adalah dengan mengunduh *source-code Mininet* pada github *Mininet* dengan perintah berikut pada terminal:

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

2. Selanjutnya untuk memulai proses instalasi *Mininet* dapat dilakukan dengan menjalankan perintah berikut:

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

3. Untuk memeriksa keberhasilan instalasi, dapat dilakukan dengan menjalankan perintah berikut :

```
$ sudo mn --test pingall
```

4.2.1.2 *Ryu Controller*

Untuk instalasi Ryu sebagai *controller* pda jaringan *Software Defined Network* dapat dengan menjalankan beberapa perintah berikut ini:

1. Langkah pertama untuk instalasi Ryu adalah dengan menginstal pip terlebih dahulu. Pip merupakat paket dari bahasa pemrograman *Python* yang nantinya akan diguanakan untuk membuat program sistem *routing*. Dapat dilakukan dengan menjalankan perintah seperti berikut pada terminal Ubuntu:

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

2. Langkah berikutnya, mengunduh *source code* ryu pada github ryu dengan menjalankan perintah seperti berikut :

```
$ git clone git://github.com/osrg/ryu.git
```

3. Selanjutnya dengan menjalankan perintah berikut untuk memulai proses instalasi Ryu *controller*

```
$ sudo pip install ryu
```

4.2.1.3 *NetworkX*

Untuk melakukan instalasi *Networkx* dapat ddilakukan dengan cara berikut:

1. Mengunduh *source code Networkx* pada github *Networkx* dengan melakukan perintah seperti berikut :

```
$ git clone https://github.com.networkx/networkx.git
```

2. Selanjutnya menjalankan perintah berikut untuk memulai proses instalasi *Networkx*:

```
$ python setup.py install --user
```

4.2.1.4 Pembangunan Topologi jaringan

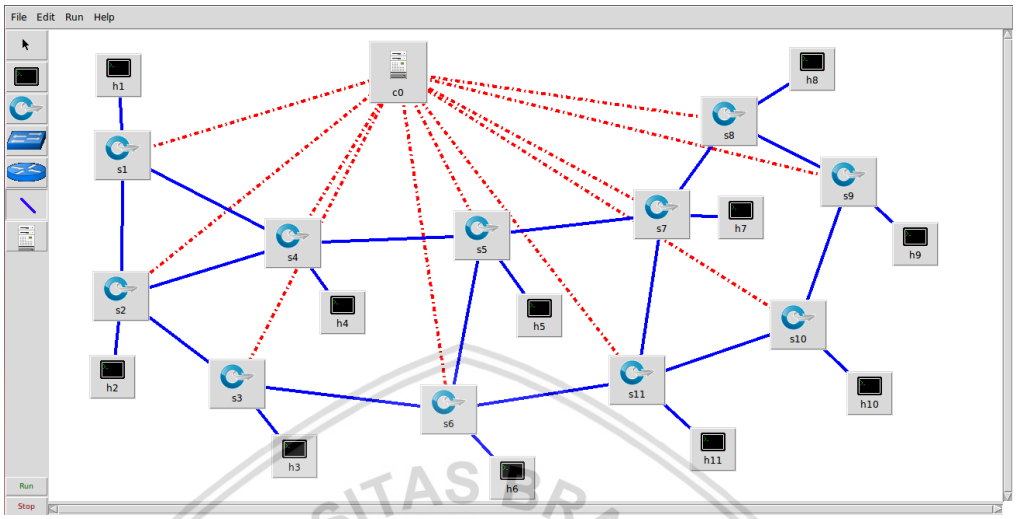
Setelah melakukan instalasi perangkat lunak pendukung sistem, maka selanjutnya pembuatan rangkaian topologi jaringan. Topologi dibangun sesuai dengan desain topologi yang telah dilakukan sebelumnya. Untuk melakukan pembangunan topologi dapat dilakukan dengan menggunakan GUI yang disebut dengan *Miniedit*.

Berikut merupakan langkah pembangunan topologi jaringan pada *Miniedit*, sebagai berikut:

1. Langkah pertama untuk menjalankan *Miniedit* ialah dengan menuliskan perintah berikut pada terminal :

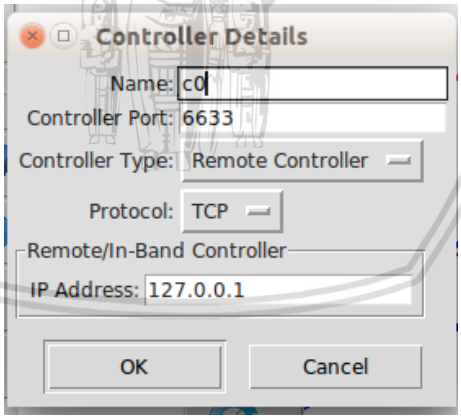
```
$ sudo mininet/examples/miniedit.py
```

2. Langkah kedua membangun topologi dapat dilakukan dengan melakukan *drag and drop* pada komponen yang diperlukan pada jaringan. Berikut merupakan contoh pembangunan topologi pada *miniedit*.



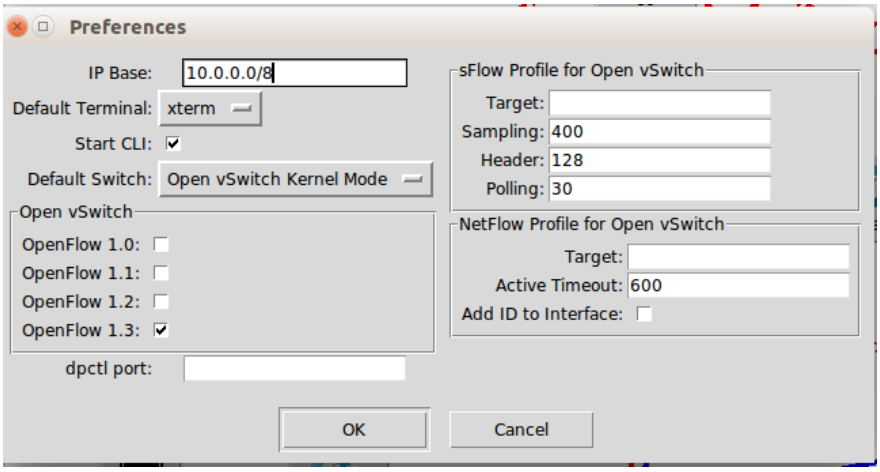
Gambar 4. 3 Topologi *Abilene*

3. Setelah pembangunan topologi jaringan selesai, kemudian dilakukan pengaturan pada *controller* (c0) dengan cara klik kanan pada *icon controller* (c0) lalu memilih menu *properties*. Kemudian mengubah "*controller type*" menjadi "*Remote Controller*" Pengaturan *controller* dilakukan dengan tujuan menghubungkan *controller* tersebut dengan *mininet*. Seperti gambar berikut:



Gambar 4. 4 Pengaturan *Controller Details*

4. Langkah selanjutnya melakukan pengaturan pada preferences dengan cara memilih Edit pada menu *toolbar miniedit* kemudian pilih *Preferences*. Setelah itu beri centang pada *OpenFlow* vesi 1.3 dan Start CLI. Seperti berikut:



Gambar 4. 5 Pengaturan *Preferences* di *Miniedit*

5. Setelah selesai semua pengaturan, langkah selanjutnya adalah menjalankan simulasi dengan cara menekan tombol "*Run*"
6. Kemudian menjalankan program *controller* dengan membuka terminal baru dan melakukan perintah seperti berikut :

```
$ sudo ryu-manager [nama_program].py –observe-links
```

4.3 Implementai Algoritme

Implementasi algoritme *routing* dalam sistem dilakukan dengan menggunakan *mininet*. Tujuan dari penerapan algoritme *routing* ini adalah untuk menuntun paket data dari node sumber agar sampai menuju node tujuan dengan rute yang paling pendek. Algoritme *routing* ini diimplementasikan pada *layer controller. Layer controller* bertugas sebagai pengontrol rute paket data yang dikirimkan. Berikut penerapan algoritme *routing* dalam jaringan *Software Defined Network*.

Implementasi Algoritme ini menjelaskan mengenai program program Algoritme *Dijkstra, Bellman-Ford* dan *Floyd-Warshall* pada jaringan *Software Defined Network* (SDN). Masing-masing algoritme memiliki mekanisme pencarian jalur yang berbeda. Pada Algoritme *Dijkstra*, mekanisme pencarian dengan menghitung seluruh *cost* yang berhubungan dengan *link* pada setiap rute untuk mendapatkan metric rute-rute yang terhubung. Sedangkan Algoritme *Bellman-Ford* memiliki mekanisme pencarian dengan menghitung *cost* minimum dari *switch* awal ke tetangganya ditambah jarak tetangganya menuju *switch* tujuan. Mekanisme pencarian Algoritme *Floyd-Warshall* melakukan pemecahan masalah dengan memandang solusi yang akan diperoleh sebagai suatu keputusan yang saling terkait. Yang artinya Algoritme *Floyd-Warshall* mengacu pada kesimpulan-kesimpulan sebelumnya.

4.3.1 Implementasi Algoritme *Djikstra*

Implementasi Algoritme *Dijkstra* akan membahas tentang Algoritme *Dijkstra* untuk diimplementasikan ke dalam program simulasi jaringan *Software Defined Network.*

```
Algoritme 1: Dijkstra
1  Def single_source_dijkstra (G, source, target=None, cutoff=None,
2       weight='weight'):
3      if source == target:
4          return ({source: 0}, {source: [source]})
5      if G.is_multigraph():
6          get_weight = lambda u, v, data: min(
7              eattr.get(weight, 1) for eattr in data.values())
8      else:
9          get_weight = lambda u, v, data: data.get(weight, 1)
10     paths = {source: [source]}  # dictionary of paths
11     return _dijkstra(G, source, get_weight, paths=paths,
12      cutoff=cutoff,
13                       target=target)
```

Algoritme 1 merupakan *source code* dari Algoritme *Dijkstra*. Berikut merupakan penjelasan dari *source code*:

Baris 1-2 merupakan untuk menemukan jalur terpendek pada G dari node sumber

Baris 3-9 menjelaskan pencarian target node yang di tuju dengan jarak terpendek

Baris 10-12 merupakan pembentukan jalur terpendek yang telah ditentukan

4.3.2 Algoritme *Bellman-Ford*

```
Algoritme 2: Bellman-Ford
1  def bellman_ford_predecessor_and_distance(G, source,
2       target=None,
3                                            cutoff=None,
4       weight='weight'):
5      if source not in G:
6          raise nx.NodeNotFound("Node %s is not found in the
7       graph" % source)
8      weight = _weight_function(G, weight)
9      if any(weight(u, v, d) < 0 for u, v, d in
10      nx.selfloop_edges(G, data=True)):
11         raise nx.NetworkXUnbounded("Negative cost cycle
12      detected.")
13     dist = {source: 0}
14     pred = {source: [None]}
15     if len(G) == 1:
16         return pred, dist
17
18     weight = _weight_function(G, weight)
19
20     dist = _bellman_ford(G, [source], weight, pred=pred,
21      dist=dist,
22                          cutoff=cutoff, target=target)
23     return (pred, dist)
```

Algoritme 2 merupakan *source code* dari Algoritme *Bellman-Ford*. Berikut merupakan penjelasan dari *source code*:

Baris 1-2 merupakan untuk menemukan jalur terpendek pada G dari node sumber

Baris 5-7 menjelaskan pencarian target node yang di tuju dengan jarak terpendek

Baris 10-12 merupakan pembentukan jalur dengan melakukan looping untuk menentukan node selanjutnya yang memiliki nilai kecil

Baris 13-14 merupakan inisialisasi variabel *dist* dan *pred*

Baris 20 merupakan menentukan tujuan dengan format

4.3.3 Algoritme *Floyd-warshall*

Algoritme 3: Floyd-Warshall

```
1  def floyd_warshall_predecessor_and_distance(G, weight='weight'):
2      from collections import defaultdict
3      dist = defaultdict(lambda: defaultdict(lambda: float('inf')))
4      for u in G:
5          dist[u][u] = 0
6      pred = defaultdict(dict)
7      undirected = not G.is_directed()
8      for u, v, d in G.edges(data=True):
9          e_weight = d.get(weight, 1.0)
10         dist[u][v] = min(e_weight, dist[u][v])
11         pred[u][v] = u
12         if undirected:
13             dist[v][u] = min(e_weight, dist[v][u])
14             pred[v][u] = v
15     for w in G:
16         for u in G:
17             for v in G:
18                 if dist[u][v] > dist[u][w] + dist[w][v]:
19                     dist[u][v] = dist[u][w] + dist[w][v]
20                     pred[u][v] = pred[w][v]
21     return dict(pred), dict(dist)
22
23
```

Algoritme 3 adalah *source code Floyd-Warshall*. Berikut merupakan penjelasan dari *source code*:

Baris 1-2 menjelaskan bagaimana menemukan semua jalur terpendek dengan menggunakan Algoritme *Floyd-Warshall*.

Baris 3-6 menjelaskan representasi kamus-kamus untuk *dist* dan *pred*, dengan menggunakan beberapa *defaultdict*. Untuk *dist default* adalah nilai *floating point inf*

Baris 7-21 merupakan inisialisasi *path distance* menjadi *matrix* kedekatan *path* dan juga mengatur jarak ke node itu sendiri menjadi 0 (nol)

BAB 5 PENGUJIAN DAN ANALISIS

Pada bab pengujian ini menjelaskan mengenai setiap proses pembangunan sistem berdasarkan skenario pengujian yang telah dilakukan sebelumnya. Pengujian dilakukan dengan menilai seuruh kebutuhan yang telah dispesifikasikan sebelumnya telah terpenuhi. Pada subab analisis ini dijelaskan mengenai analisis dari pengujian yang dilakukan pada bab sebelumnya. Analisis yang dilakukan bertujuan untuk menilai sistem bekerja dengan sesuai yang diharapkan. Setelah dilakukannya analisis, maka akan diperoleh hasil sebagai acuan untuk menarik kesimpulan dari penelitian ini.

5.1 Pengujian

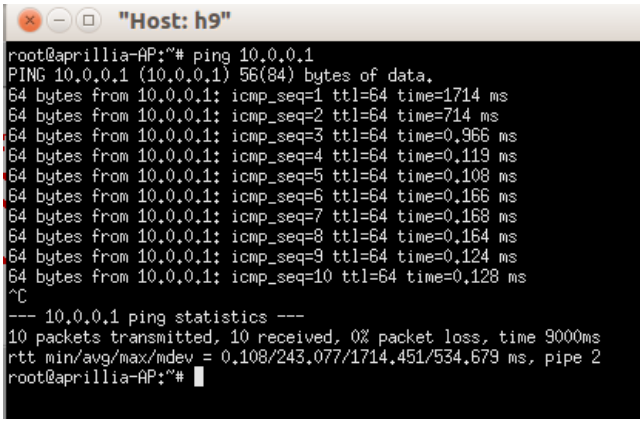
Pada sub bab pengujian ini dilakukan untuk mempermudah melakukan analisis dan melakukan perbandingan hasil performa dari Algoritme *Dijkstra, Bellman Ford* dan *Floyd Warshall* pada jaringan *Software Defined Network*. Pengujian dilakukan dengan menggunakan topologi Abilene. Nilai *cost* antar *switch* ditentukan dengan inputan manual dengan besaran Mbps (*Mega bit per second)*. *Cost link* akan ditentukan dengan mengatur *link bandwidth* pada setiap *link* topologi simulasi jaringan *Software Defined Network* yang akan menggunakan 3 jenis besaran *link bandwidth* yaitu 10 Mbps, 100 Mbps dan 1000 Mbps. Hal ini dimaksudkan untuk menambah kompleksitas dari sebuah jaringan. Penghitungan nilai *cost* didapat dengan rumus berikut :

$$Cost = Reference\ Bandwidth \div Link\ Bandwidth$$

Pengujian dilakukan dengan melakukan penilaian terhadap hasil dari tiap nilai yang dihasilkan dari penerapan algoritme *routing* pada sistem simulasi. Pengambilan hasil dari pengujian ketiga algoritme *routing* akan dibandingkan dan dianalisis dengan mengacu pada parameter uji *convergence time, throughput, Recovery time* dan *packet loss.* Pengujian pemilihan jalur menggunakan tools aplikasi bwm-ng. Bwm-ng merupakan tools yang digunakan untuk mengetahui *throughput* pada *outport switch*.

5.1.1 Pengujian Validasi

Pada pengujian validasi dilakukan untuk mengetahui apakah program yan dibuat sesuai dengan yang diharapkan. Pengujian ini dilakukan dengan mengirimkan paket ICMP dari host sumber (*source*) ke host tujuan (*destination*). Dengan menentukan host sumber dan host tujuan akan dilakukan penghitungan secara manual dengan menggunakan logika algoritme *Dijkstra, Bellman-Ford* dan *Floyd-Warshall*. Pengiriman paket ICMP dilakukan dengan menuliskan *command* "ping [IP destination]" seperti gambar 5.1 berikut. Kemudian membandingkan dengan hasil *path* yang dihasilkan dari masing-masing program. Langkah selanjutnya ialah menganalisis hasil dari masing-masing algoritme, apakah jalur yang didapatkan oleh masing-masing algoritme pada program sesuai dengan penghitungan manual.



Gambar 5. 1 Pengiriman Paket dari host 9 ke host 1

Gambar 5.1 merupakan gambar pengiriman paket ICMP dari host 9 yang bertindak sebagai host sumber (*source*) ke host 1 yang bertindak sebagai host tujuan (*destination*).

5.1.1.1 Algoritme *Dijkstra*

Algoritme *Dijkstra* melakukan pencarian jalur terpendek dengan jumlah *cost* yang paling kecil. Berikut adalah rumus dari algoritme *Dijkstra* :

$$D_{(v)} = \min \ (D_v, M_v + E_w)$$

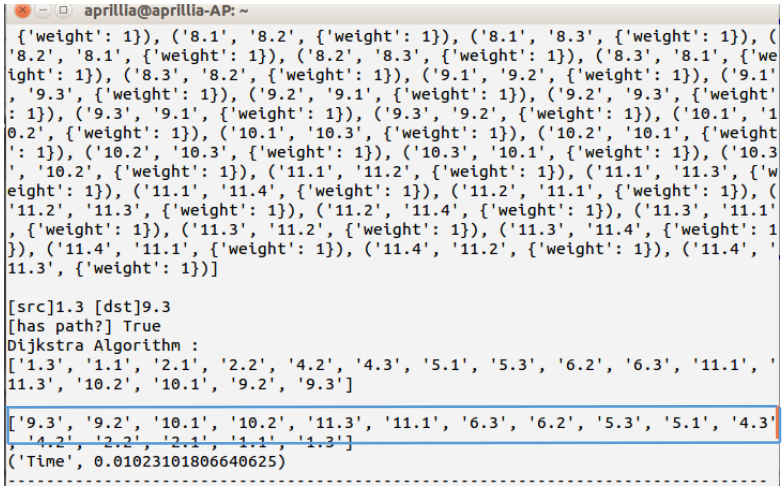
Berdasarkan persamaan di atas D(v) adalah *DestValue* yang merupakan nilai dalam *verteks* tujuan, Mv (*MarkedValue*) yang mengindikasika nilai dalam *vertex* awal, dan Ew (*EdgeWeight*) adalah bobot dari sisi yang menghubungkan *vertex*. Berikut merupakan mekanisme algoritme *Dijkstra* menentukan jalur terpendek dengan penghitungan manual.

step	N'	s9	s10	s11	s8	s7	s6	s5	s4	s3	s2	s1	h1
1	h9	0,h9	~	~	~	~	~	~	~	~	~	~	~
2	h9-s9		1,s9	~	100,s9	~	~	~	~	~	~	~	~
3	h9-s9-s10			2,s10	100,s9	~	~	~	~	~	~	~	~
4	h9-s9-s10-s11					102,s11	12,s11	~	~	~	~	~	~
5	h9-s9-s10-s11-s6							13,s6	~	32,s6	~	~	~
6	h9-s9-s10-s11-s6-s5								14,s5		~		~
7	h9-s9-s10-s11-s6-s5-s4										24,s4	114,s4	~
8	h9-s9-s10-s11-s6-s5-s4-s2											34,s2	~
9	h9-s9-s10-s11-s6-s5-s4-s2-s1-h1												34,s1-h1
10													

Gambar 5. 2 Hitungan Manual Algoritme *Dijkstra*

Berdasarkan hasil penghitungan manual pada gambar 5. Menunjukkan bahwa jalur terpendek dari host 9 menuju host 1 adalah dengan melalui s9-s10-s11-s6-s5-s4-s2-s1 dengan jumlah *cost* 34.

Gambar 5.3 berikut merupakan *screenshot* dari hasil keluaran pengiriman paket dengan menggunakan algoritme *Dijkstra*. Dapat dilihat dari gambar Tersebut terdapat barisan angka yang menunjukkan jalur yang dilalui oleh paket dari host 9 ke host 1. Jalur terpendek yang dilalui paket oleh algoritme *Dijkstra* yaitu melalui *switch* 9-10-11-6-5-4-2-1.



Gambar 5. 3 Jalur yang dilalui dengan Algoritme *Dijkstra*

5.1.1.2 Algoritme *Bellman-Ford*

Algoritme *Bellman-Ford* menentukan jalur terpendek dengan melakukan publikasi tabel *routing* yang dimiliki ke tetangga. Mekanisme *routing Bellman-Ford* adalah dengan memberikan setiap *router* informasi tabel *routing* sehingga dapat diketahui jalur ke setiap *switch* dan lintasan yang akan dipakai dalam pengiriman paket menuju tujuan. Berikut adalah rumus algoritme *Bellman-Ford*:

$$D_x(y) = \min\{c(x,y) + d_v(y)\}$$

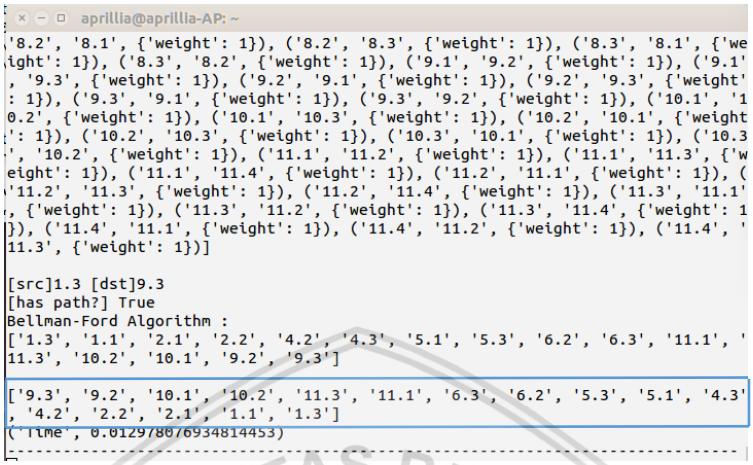
Berdasarkan persamaan di atas $dx(y)$ adalah jarak dari *switch* x ke y. $c(x, v)$ merupakan *cost* dari *switch* x ke v. Sedangkan $dv(y)$ adalah jarak v ke y. Berikut merupakan penghitungan manual untuk menentukan jalur terpendek dari host 1 ke host 9.

	iterasi 1
ds9(h1)	Min{c (s9,s8) + ds8(H1), c (s9,s10) + ds10(H1)}
	Min { c (100+32) , c (1+33)}
	Min { c (132) , c (34)}
	Min (34) **s10**
	iterasi 2
ds10(h1)	Min{c (s10,s11) + ds11(H1)}
	Min { c (1+32) }
	Min { c (33)}
	Min (33) **s11**
	iterasi 3
ds11(h1)	Min{c (s11,s7) + ds7(H1), c (s11,s6) + ds6(H1)}
	Min{c (100 + 31), (10+22)}
	Min (32) **s6**
	iterasi 4
ds6(h1)	Min{c (s6,s5) + ds5(H1), c (s6,s3)+ds3(h1)}
	Min { c (1+21), (10 + 110) }
	Min { c (22) , c (120)}
	Min (22) **s5**

	iterasi 5
ds5(h1)	Min{c (s5,4) + ds4(H1)
	Min{c (1) + (20)}
	Min (21) **s4**
	iterasi 6
ds4(h1)	Min{c (s4,s1) + ds1(H1), c (s2,s1) + ds1(H1)}
	Min { c (100) , c (10+10)}
	Min { c (100) , c (20)}
	Min (20) **s2**
	iterasi 7
ds2(h1)	Min{c (s2,s1) + ds1(H1)}
	Min{c (10))}
	Min (10) H1
	Path yang terbentuk :
	H9-S9-S10-S11-S6-S5-S4-S2-S1-H1
	cost = 34

Gambar 5. 4 Hitungan Manual Algoritme *Bellman-Ford*

Gambar di atas merupakan perhitungan manual *Bellman-Ford*. Menunjukkan hasil yang sama seperti perhitungan manual algortime *Dijkstra* bahwa jalur terpendek dari host 1 menuju host 9 adalah dengan melalui s1-s2-s4-s5-s6-s11-s10-s9 dengan jumlah *cost* 34.



Gambar 5. 5 Jalur yang dilalui dengan Algoritme *Bellman-Ford*

Gambar di atas merupakan *screenshot* dari hasil keluaran pengiriman paket dengan menggunakan algoritme *Bellman-Ford*. Dapat dilihat dari gambar 5. Tersebut terdapat barisan angka yang menunjukkan jalur yang dilalui oleh paket dari host 9 ke host 1. Jalur terpendek yang dilalui paket oleh algoritme *Bellman-Ford* yaitu melalui *switch* 9-10-11-6-5-4-2-1.

5.1.1.3 Algoritme *Floyd-Warshall*

Algoritme *Floyd-Warshall* menentukan jalur terpendek dengan membandingkan semua lintasan yang mungkin dalam graf untuk setiap pasang simpul. Berikut merupakan rumus dari algoritme *Floyd-Warshall*:

$$d_{ij}{}^{(k)} = \min\{d_{ij}{}^{(k-1)}, d_{ik}{}^{(k-1)} + d_{kj}{}^{(k-1)}$$

d_{ij} merupakan jarak dari vertex i ke j. Untuk sebuah shortest path dari i ke i dengan beberapa vertex intermediate pada path dipilih dari kumpulan {1,2,..k} dengan 2 kemungkinan. Yang pertama k merupakan pat terpendek yang mempunyai panjang $d_{ij}{}^{(k-1)}$. Atau k adalah vertex pada path terpendek mempunyai panjang $d_{ik}{}^{(k-1)} + d_{kj}{}^{(k-1)}$. Berikut merupakan perhitungan manual algoritme *Floyd-warshall*.

Berikut merupakan perhitungan manual dari pencarian jalur algoritme *Floyd-Warshall.* Terdapat 11 iterasi:

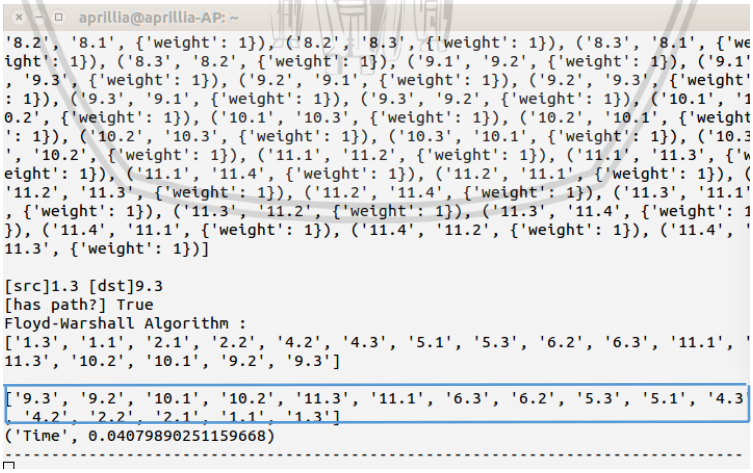
D(0)	1	2	3	4	5	6	7	8	9	10	11
1	0	10	~	100	~	~	~	~	~	~	~
2	10	0	100	10	~	~	~	~	~	~	~
3	~	100	0	~	~	~	~	~	~	~	~
4	100	10	~	0	1	~	~	~	~	~	~
5	~	~	~	1	0	1	~	~	~	~	~

6	~	~	10	~	1	0	~	~	~	~	10
7	~	~	~	~	10	~	0	1	~	~	100
8	~	~	~	~	~	~	1	0	100	~	~
9	~	~	~	~	~	~	~	100	0	1	~
10	~	~	~	~	~	~	~	~	1	0	1
11	~	~	~	~	~	10	100	~	~	1	0

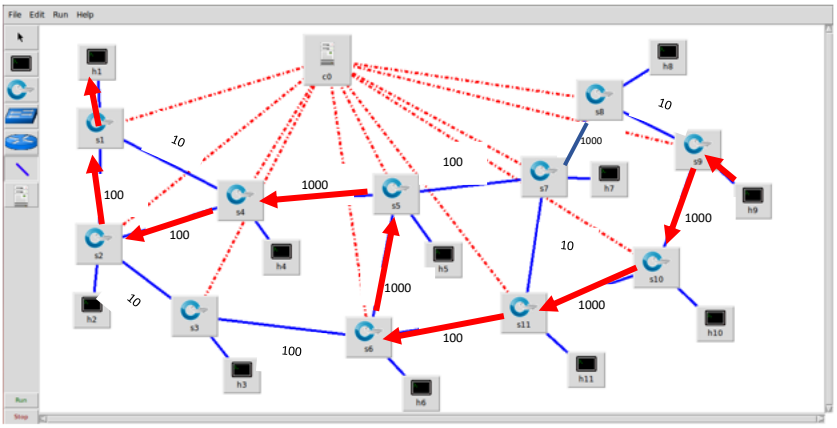
D(11)	1	2	3	4	5	6	7	8	9	10	11
1	0	10	32	20	21	22	132	133	34	33	32
2	10	0	22	10	11	12	122	123	24	23	22
3	32	22	0	12	11	10	120	121	22	21	20
4	20	10	12	0	1	2	112	113	14	13	12
5	21	11	11	1	0	1	111	112	13	12	11
6	22	12	10	2	1	0	110	121	12	11	10
7	132	122	120	112	111	110	0	1	101	101	100
8	133	123	121	113	112	111	1	0	100	101	101
9	34	24	22	14	13	12	101	100	0	1	2
10	33	23	21	13	12	11	101	101	1	0	1
11	32	22	20	12	11	10	100	101	2	1	0

Gambar 5. 6 gambar tabel iterasi 0 dan 11 penentuan jalur algoritme *Floyd-Warshall*

Gambar berikut merupakan *screenshot* dari hasil keluaran pengiriman paket dengan menggunakan algoritme *Floyd-Warshall*. Dapat dilihat dari gambar 5.7 Tersebut terdapat barisan angka yang menunjukkan jalur yang dilalui oleh paket dari host 9 ke host 1. Jalur terpendek yang dilalui paket oleh algoritme *Floyd-Warshall* yaitu melalui *switch* 9-10-11-6-5-4-2-1.



Gambar 5. 7 Jalur yang dilalui dengan Algoritme *Floyd-Warshall*



Gambar 5. 8 Jalur yang Dilalui Paket ICMP dari host 1 ke host9

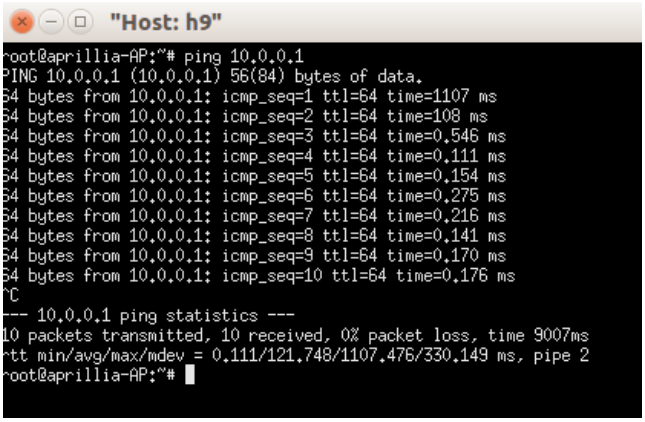
Gambar 5.8 merupakan topologi jaringan *Abilene* yang digunakan dalam penelitian ini. Pada gambar di atas terlihat arah panah warna merah yang menunjukkan alur paket yang dikirimkan. Nilai *cost* antar *switch* ditentukan dengan inputan manual dengan besaran Mbps (*Mega bit per second)*. *Cost link* akan ditentukan dengan mengatur *link bandwidth* pada setiap *link* topologi. Cost tiap link didapatkan dari pembagian antara reference bandwidth dan link bandwidth. Nilai dari reference bandwidth yang digunakan adalah 1000 Mbps dan untuk link bandwidth peneliti menggunakan 3 jenis besaran kapasitas yaitu 10 Mbps, 100 Mbps dan 1000 Mbps yang ditunjukkan dengan angka berwarna hitam pada gambar topologi di atas. Maka semakin besar *bandwidth* yang terdapat pada *link* nilai *cost* antar *switch*pun semakin kecil.

5.1.2 Pengujian *Convergence Time*

Pengujian *Convergence Time* dilakukan dengan tujuan mengetahui berapa waktu yang diperlukan oleh *controller* untuk mendapatkan jalur yang diperlukan untuk mengirimkan paket dengan jumlah *cost* paling kecil. Penghitungan *Convergence Time* dilihat dari ketika *switch* menerima paket dan meminta tabel *routing* kepada *controller* hingga *controller* memberikan table *routing* kepada *switch*.

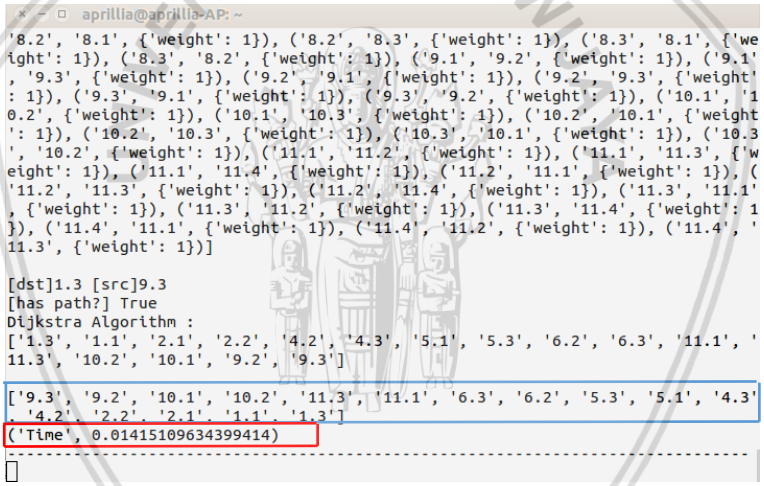
Pada pengujian *convergence time* dilakukan dengan cara mengirimkan paket ICMP dari host sumber ke host tujuan dengan menggunakan *ping*. Kemudian *timer* pada program *controller* akan berjalan menghitung waktu proses algoritme tersebut. Setelah proses pencarian jalur terpendek selesai, maka *controller* akan menampilkan jalur terpendek yang dilaui oleh paket dan menampilkan waktu yang menunjukkan *convergence time*.

5.1.2.1 Pengujian *Convergence Time* Algoritme *Dijkstra*



Gambar 5. 9 Hasil ping dari host 1 ke host 11 pada Algoritme *Dijkstra*

Pada gambar 5.9 merupakan *screenshot* dari host 9 sebagai *client* yang mengirimkan paket ICMP menuju host 1 sebagai *server* untuk memicu *controller* menjalankan algoritme pencarian jalur. Kemudian *controller* akan memulai *timer* yang digunakan sebagai perhitungan *convergence time*.



Gambar 5. 10 *Output* pengujian *Convergence Time* Algoritme *Dijkstra*

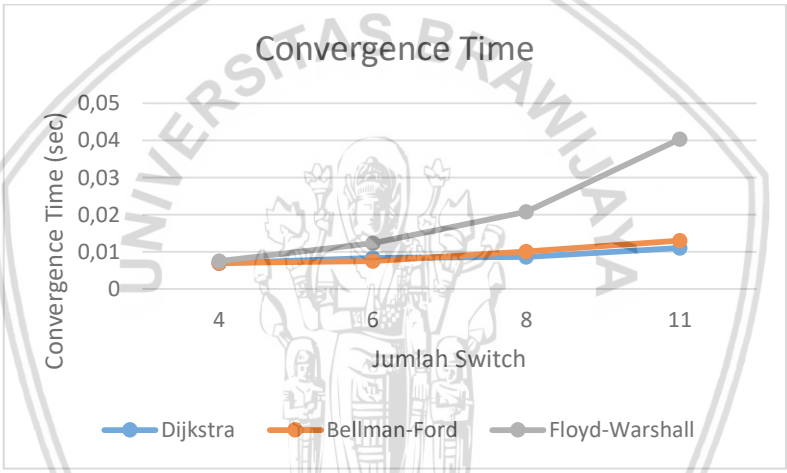
Pada gambar 5.10 di atas merupakan hasil *output* dari *convergence time* yang dilakukan dengan menggunakan Algoritme *Dijkstra*. Hasil *routing* yang diperoleh merupakan jalur terpendek yang dilalui paket dari host 9 ke host 1. Dari gambar di atar menujukkan jalur yang dilalui yaitu *switch* 9-10-11-6-5-4-2-1. Waktu yang didapatkan untuk pengujian *convergence time* yaitu selama 0,0141 detik.

Untuk mendapatkan nilai *convergence time* pengujian dilakukan dengan mengirimkan paket ICMP dari host sumber ke host tujuan. Pengujian ini dilakukan dengan variasi pembeda yaitu jumlah *switch* yang berbeda dengan tujuan untuk mengetahui perubahan jumlah *switch* pada topologi berpengaruh pada waktu tempuh *controller* untuk mendapatkan jalur terpendek. Untuk melakukan pengujian ini digunakan *tools ping* dengan *command* pada host sumber “ping [host tujuan]”.

Tabel 5. 1 Hasil Pengujian Convergence Time

JUMLAH SWITCH	*Convergence Time (sec)*		
	Rata-rata Dijkstra	Rata-rata Bellman	Rata-rata Floyd-Warshall
4	0.006942	0.00698	0.007514
6	0.008437	0.007571	0.012325
8	0.00864	0.010094	0.020827
11	0.011086	0.013088	0.04037

Tabel 5.1 merupakan hasil pengujian *convergence time* yang menunjukkan nilai dari pengujian convergence time dilakukan dengan 10 percobaan pada setiap algoritme *Dijkstra, Bellman-Ford* dan *Floyd-Warshall* dengan melakukan pengiriman paket ICMP dengan kategori jumlah *switch* yang berbeda yaitu sedikit, sedang dan banyak atau dalam bentuk angka yaitu 4, 6, 8, dan 11 *switch*.

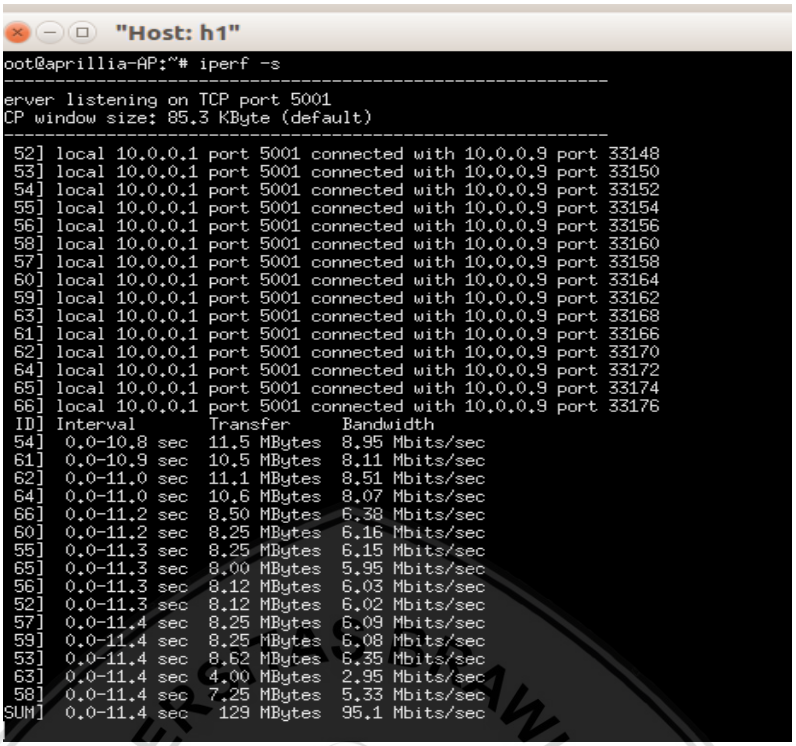


Gambar 5. 11 Grafik Hasil Pengujian Convergence Time

Pada gambar 5.11 Diatas merupakan grafik perbandingan pengujian *convergence time*. Terdapat 3 garis yang merepresentasikan *convergence time* dari masing-masing algoritme dengan variabel pembeda yaitu jumlah *switch* yang berbeda.

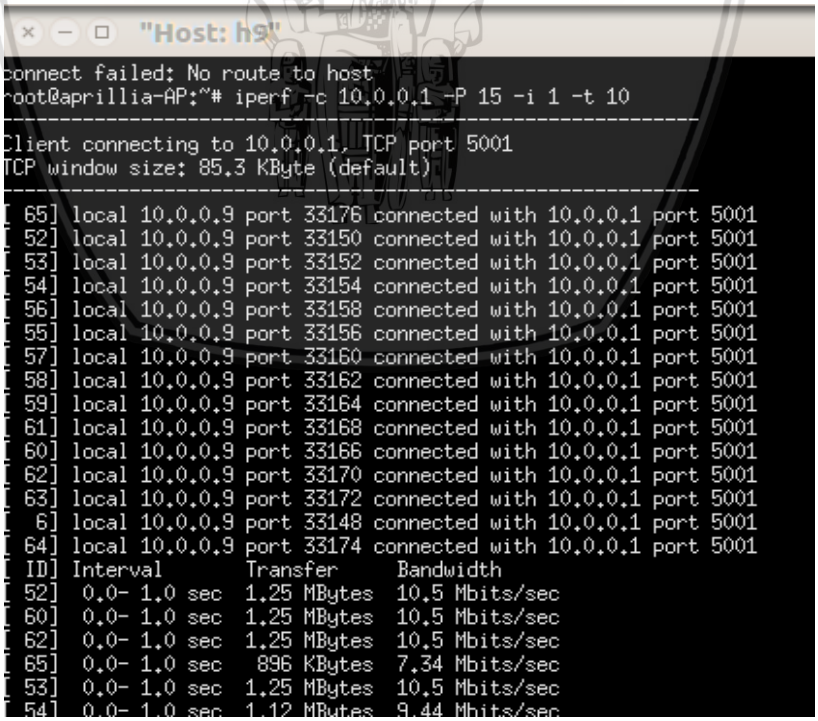
5.1.3 Pengujian *Throughput*

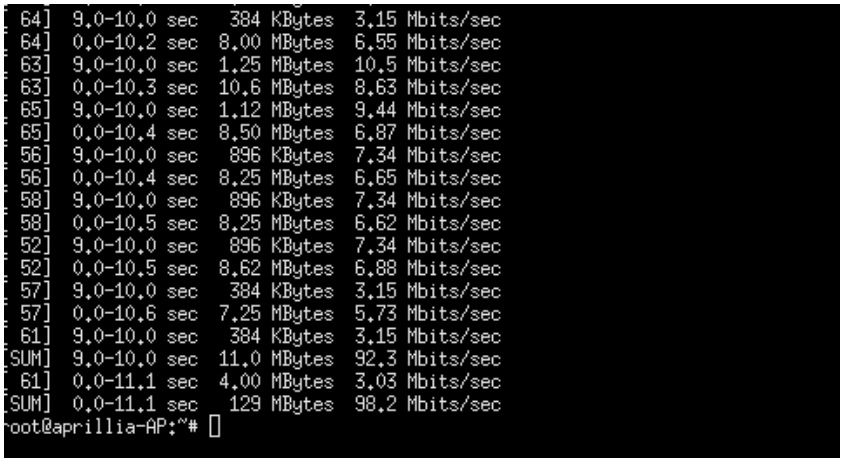
Pengujian *Throughput* dilakukan untuk mengetahui kualitas jaringan pada sistem. Pengujian dilakukan pada topologi *Abilene* yang terdapat 11 *switch* dan terdapat 1 host ditiap *switch*. Host 1 akan bertindak sebagai *server* dan host 9 akan bertindak sebagai *client* pada ujung topologi.



Gambar 5. 12 h1 berperan sebagai server

Gambar 5.12 merupakan *screenshoot* dari host 1 yang berperan sebagai *server* untuk pengiriman paket TCP dari host 9 yang berperan sebagai *client*.





Gambar 5. 13 Contoh *output* pengujian *Throughput* algoritme *Dijkstra*

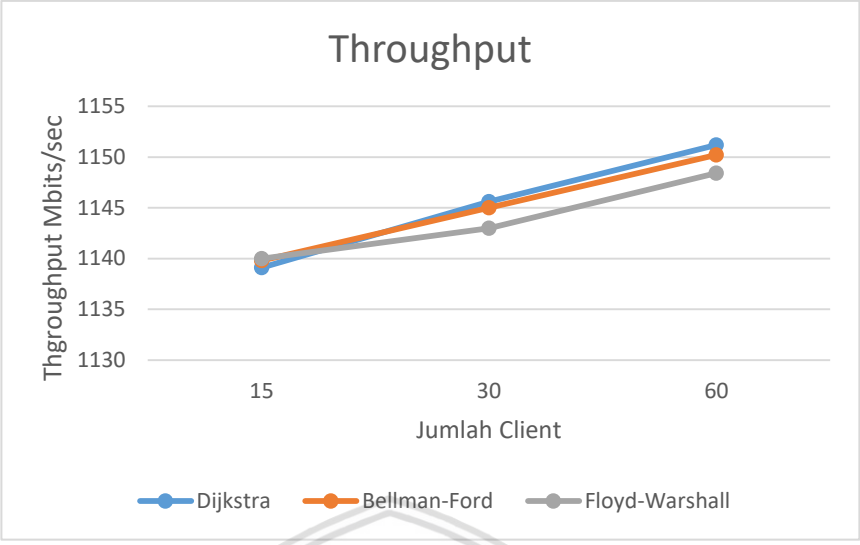
Pada gambar 5.13 di atas merupakan potongan *output* pengujian *Throughput* dengan waktu pengiriman 10 detik, nilai interval 1 dan jumlah koneksi sebanyak 15 paralel koneksi yang merupakan banyak jumlah *client* dengan menggunakan algoritme *Dijkstra* dengan nilai *Throughput* 98.2 Mbits/sec. Hal yang sama akan dilakukan pada tiap algoritme dengan 3 kategori jumlah *client* yang berbeda yaitu sedikit, sedang dan banyak. Kategori jumlah *client* dalam di representasikan dalam angka yaitu 15, 30 dan 60.

Untuk mendapatkan nilai *throughput*, *client* akan mengirimkan paket TCP menuju *server*. Pada pengujian *throughput* ini akan diberi variabel pembeda yaitu jumlah *client* yang bertujuan untuk mengetahui pengaruh dari perbedaan jumlah *client* terhadap perubahan *throughput* yang didapatkan. Pengujian dilakukan sebanyak 5 kali untuk setiap algoritme. Untuk melakukan pengujian ini digunakan tools *iperf* dengan menuliskan *command* "iperf –s" pada sisi host yang berperan sebagai *server* dan menuliskan *command* "iperf –c [ip server] [parameter]" pada sisi host yang berperan sebagai *client*. Parameter yang digunakan pada client untuk mengirimkan data yaitu memiliki durasi pengiriman selama 10 detik, memiliki waktu interval selama1 detik dan jumlah *client* pada host yang bertindak sebagai *client* yang terhubung ke *server*. Hasil dari pengujian *throughput* tersaji dalam tabel dan gambar berikut:

Tabel 5. 2 Tabel Hasil Pengujian *Throughput*

Jumlah Client	Rata –rata Throughput Mbps/sec		
	Dijkstra	Bellman-Ford	Floyd-Warshall
15	1139,1	1139,8	1140
30	1145,6	1145	1143
60	1150,8	1150,2	1148,4

Pada tabel 5.2 di atas merupakan hasil pengujian *throughput* pada topologi *Abilene*. Variabel perbedaan jumlah *client* sebanyak 15, 30 dan 60.

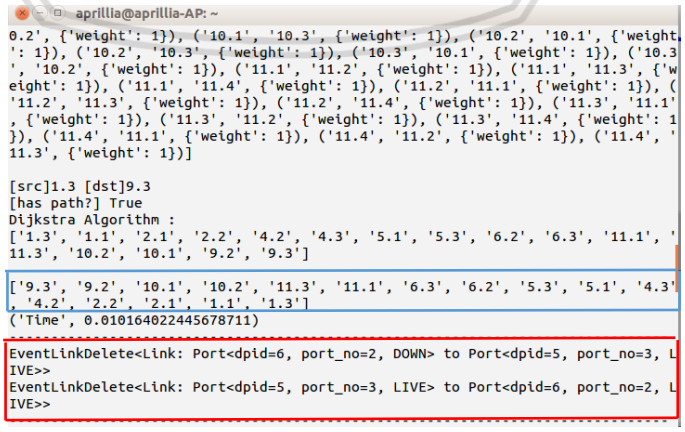


Gambar 5. 14 Grafik Hasil Pengujian *Throughput*

Gambar 5.14 diatas merupakan grafik perbandingan hasil pengujian *throughput*. Terdapat 3 garis yang merepresentasikan *throughput* dari masing-masing algoritme dengan variabel pembeda yaitu jumlah host *client* yang berbeda.

5.1.4 Pengujian *Recovery Time (Link Failure)*

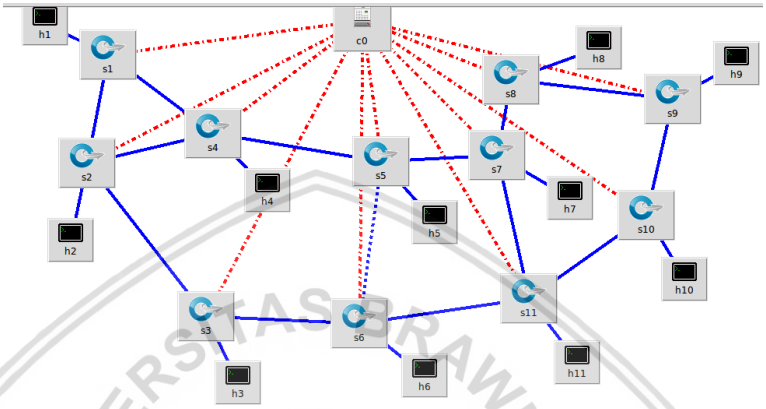
Pengujian ini menghitung berapa lama waktu yang dbutuhkan oleh suatu sistem jaringan untuk kembali beroperasi setelah mengalami masalah dan keadaan dimana sistem tersebut mengalami *failure*, sistem akan mengukur berapa lama waktu yang dibutuhkan oleh jaringan tersebut hingga beroperasi normal. Pemutusan suatu link atau *link failure* yang ada pada topologi dan melihat seberapa cepat *recovery time* yang diperlukan oleh *controller* untuk membentuk jalur terpendek baru dari tiga algoritme *routing* yang telah diterapkan pada jaringan *Software defined network* yaitu algoritme *Dijkstra, Bellman-Ford* dan *Floyd-Warshall*. Pada aplikasi *Miniedit Link/*jalur yang diputus akan terlihat garis putus-putus.



Gambar 5. 15 Jalur dan *Convergence Time* pada pertama kali pencarian

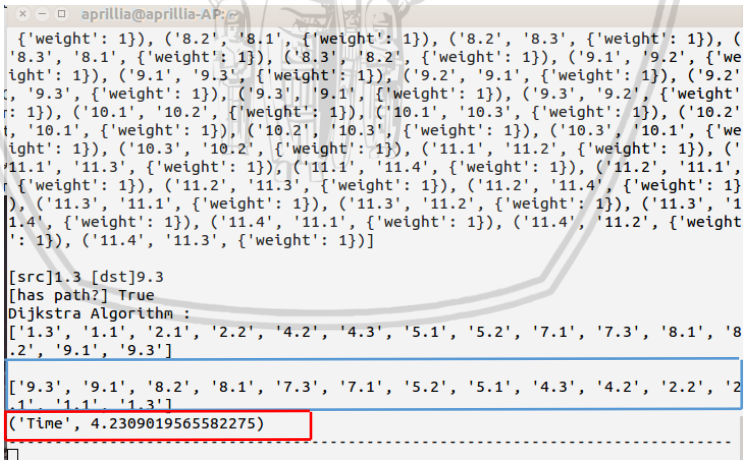
Gambar 5.15 menunjukkan jalur yang dilalui pertama kali oleh paket dari host 9 ke host 1 dengan melewati *switch* 9-10-11-6-5-4-2-1 menggunakan algoritme

Dijkstra dan terdapat *convergence time* yaitu selama 0.010 detik. Pada kotak berwarna merah menunjukkan bahwa *link* antara *switch* 5 ke 6 telah terhapus atau mengalami *link failure*. Pengujian dilakukan dengan melakukan pengiriman paket ICMP dengan perintah "*ping* [IP_server]" sama seperti pengujian *convergence time*. Setelah pengiriman berhasil dan menghasilkan jalur pengiriman, akan dilakukan pemutusan salah satu *link*. Dari pemutusan link tersebut *controller* akan mencari jalur baru untuk mengirimkan paket dan waktu yang dibutuhkan untuk membentuk jalur baru tersebut merupakan *Recovery Time*.



Gambar 5. 16 Topologi yang telah mengalami *link failure*

Gambar 5.16 merupakan topologi jaringan *Abilene* yang mengalami *link failure* pada jalur antara *switch* 5 dan *switch* 6 yang ditandai dengan garis putus-putus berwarna biru.



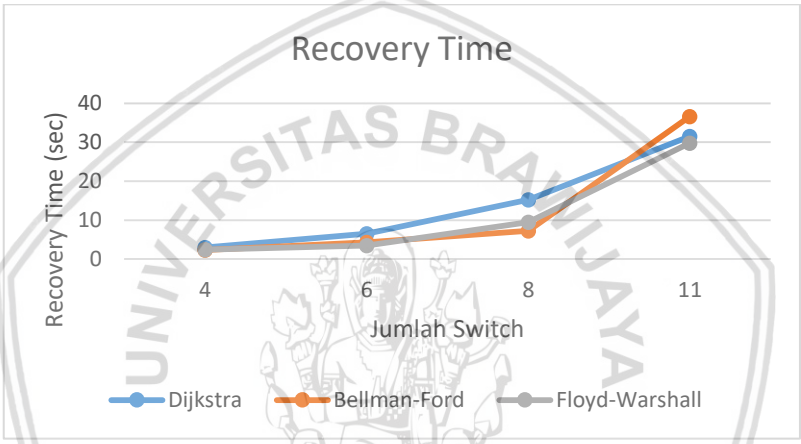
Gambar 5. 17 Jalur baru dan *Recovery Time*

Pada gambar 5.17 merupakan output pengujian *recovery time* pada algoritme *Dijkstra*. Telah dilakukan pemutusan *link* pada antara *switch* 5 dan *switch* 6 yang dapat dilihat pada gambar 5.16. Setelah pemutusan *link* maka akan ditampilkan jalur baru yaitu melalui 9-8-7-5-4-2-1 yang ditunjukkan pada kotak warna bitu dengan *recovery time* 4.23 detik yang ditunjukkan pada kotak warna merah.

Tabel 5. 3 Tabel Hasil Pengujian *Recovery Time*

JUMLAH SWITCH	Recovery Time (sec)		
	Rata-rata Dijkstra	Rata-rata Bellman	Rata-rata Floyd-Warshall
4	2.991445	2.34128	2.386603
6	6.526532	4.292443	3.462573
8	15.19746	7.295107	9.482393
11	31.51275	36.57086	29.81131

Tabel 5.3 menunjukkan bahwa pengujian ini dilakukan sebanyak 10 kali dan memiliki 3 kategori variabel pembeda yaitu sedikit, sedang dan banyak sama seperti saat pengujian *convergence time*.



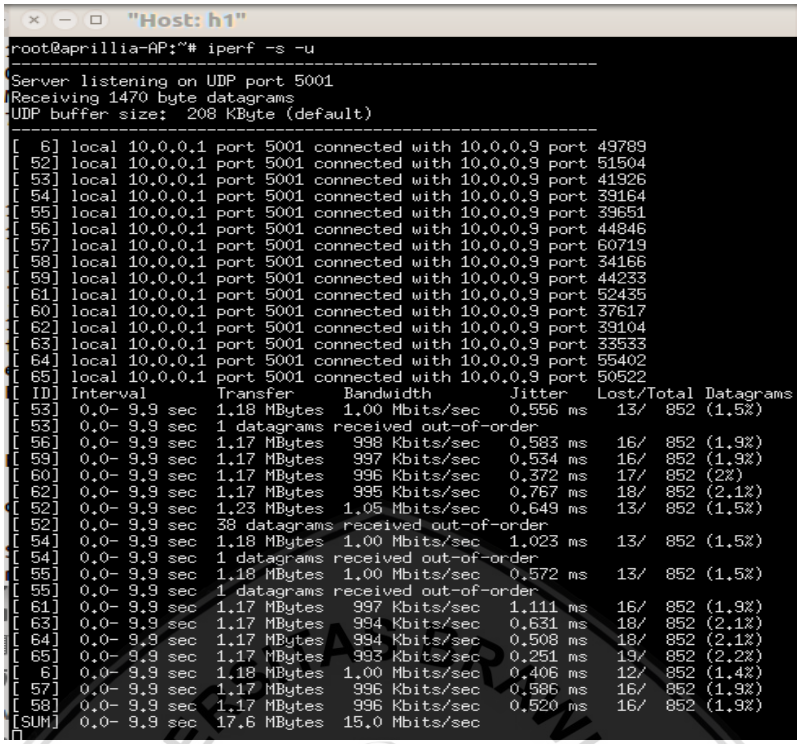
Gambar 5. 18 Tabel Hasil Pengujian *Recovery Time*

Pada gambar 5.18 merupakan grafik perbandingan pengujian *convergence time*. Terdapat 3 garis yang merepresentasikan *convergence time* dari masing-masing algoritme dengan variabel pembeda yaitu jumlah *switch* yang berbeda.

5.1.5 Pengujian *Packet Loss*

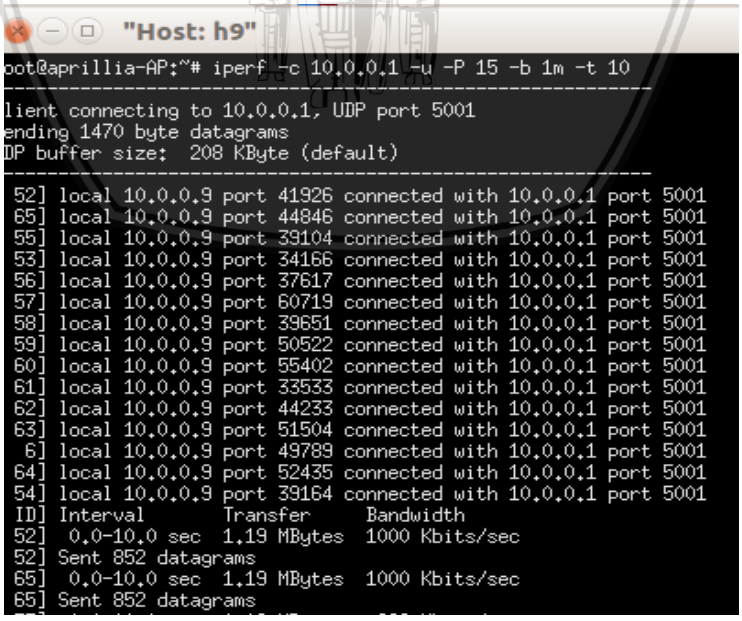
Pengujian *packet loss* dilakukan dengan tujuan untuk mengetahui kegagalan dalam mengirimkan suatu paket dari *client* ke *server*. Pengujian akan dilakukan pada masing-masing algoritme dengan 5 kali percobaan.

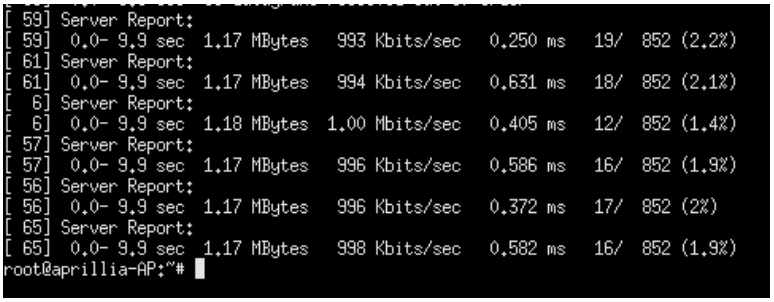
Gambar 5.19 berikut ini merupakan *screenshot* host 1 yang bertindak sebagai *server*. Untuk mendapatkan nilai *packet loss, client* akan mengirimkan paket UDP secara bersamaan dengan *rate transfer* sebesar 1 Mbits/sec atau setara dengan kualitas *youtube* tipe 480p dengan *encoding* h264 selama 10 detik. Pengujian dilakukan dengan memberikan variabel pembeda berupa kategori jumlah *client* yang berbeda yaitu sedikit, sedang dan banyak.



Gambar 5. 19 host 1 bertindak sebagai server untuk pengujian *packet loss*

Kategori jumlah *client* dalam di representasikan dalam angka yaitu 15, 30 dan 60 dengan tujuan untuk mengetahui pengaruh jumlah *client* terhadap nilai *packet loss* yang didapatkan. Pengujian menggunakan *tools iperf* dengan menuliskan *command* "*iperf -s -u*" pada sisi *server* dan pada sisi *client* dengan *command* "*iperf* –c [ip_*server*] -u -P [jumlah_*client*] –b 1m -t 10".





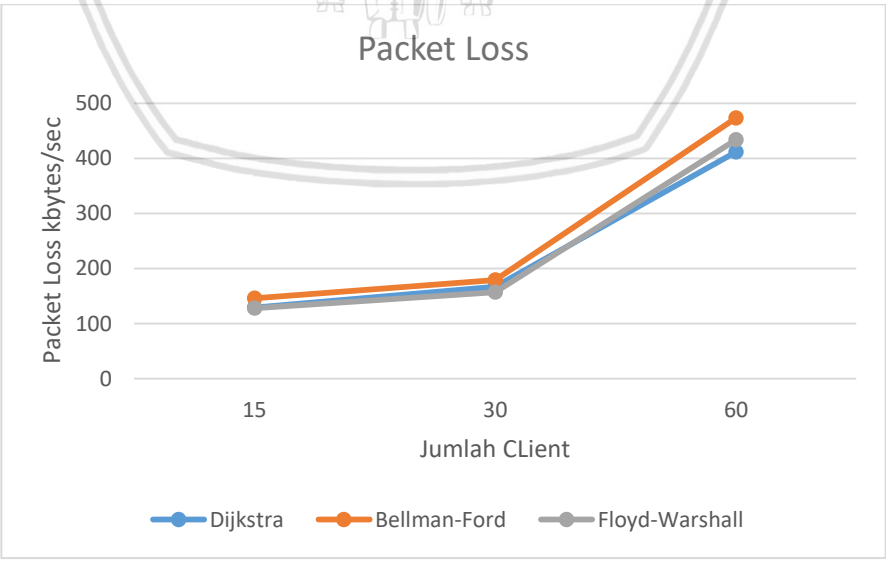
Gambar 5. 20 host 9 sebagai *client* mengirimkan paket UDP pada pengujian *packet loss*

Gambar 5.20 di atas merupakan potongan *screenshot* host 9 yang bertindak sebagai *client* mengirimkan paket UDP dengan jumlah 15 *client* secara bersamaan dengan ukuran 1 Mbits kepada *server*. dapat dilihat bahwa pengiriman paket UDP tidak sepenuhnya terkirimkan, terdapat 16 Kilobyts paket hilang.

Tabel 5. 4 Hasil Pengujian *Packet Loss*

Jumlah Client	Rata-Rata Packet Loss (kb)		
	Dijkstra	Bellman-Ford	Floyd-Warshall
15	129	146	128
30	166.6	178.8	157
60	411.2	473.2	433.4

Pada tabel diatas merupakan hasil pengujian *packet loss*. Masing-masing algoritme mendapatkan perlakukan yang sama dalam melakukan pengujian dengan variabel pembeda yaitu jumlah *client*. 3 kategori jumlah *client* yang berbeda yaitu sedikit, sedang dan banyak. Kategori jumlah *client* dalam di representasikan dalam angka yaitu 15, 30 dan 60.



Gambar 5. 21 Grafik Hasil Pengujian *Packet Loss*

Gambar 5.21s berikut merupakan grafik perbandingan hasil pengujian *packet loss*. Terdapat 3 garis yang merepresentasikan *packet loss* dari masing-masing algoritme dengan variabel pembeda yaitu jumlah *client* yang berbeda.

5.2 Analisis

Pada subab ini membahas mengenasi analisis hasil pengujian dari masing-masing algoritme *routing*. Dari hasil pengujian akan dianalisis kemudian dari hasil analisi masing algoritme akan dibandngan untuk memperoleh algoritme dengan rute terpendek yang menghasilkan nilai *cost minimum*.

5.2.1 Pengujian Validasi

Pengujian validasi dilakukan dengan menghitung secara manual segala kemungkinan sesuai skenario pengujian yang ditetapkan. Dengan pengujian validasi ini maka peneliti pengetahui program yang telah dibuat telah sesuai dengan logika algoritme yang diterapkan.

Dari hasil yang didapatkan pada pengujian validasi, pada setiap algoritme telah berhasil menentukan jalur terpendek dengan jumlah *cost* paling kecil sesuai dengan perhitungan manual. Pada *Dijkstra, Bellman-Ford* dan *Floyd-Warshall* telah menentukan jalur yang sama untuk mengirimkan paket yaitu melalui *switch* 9-10-11-6-5-4-2-1 dengan jumlah *cost* yaitu 34. Dapat disimpulkan bahwa program yang dibuat telah sesuai dengan harapan penguji dalam penentuan jalur terpendek.

5.2.2 Analisis *Convergence Time*

Convergence time merupakan waktu yang dibutuhkan oleh controller untuk membentuk sebuah jalur pengiriman paket. Berdasarkan hasil pengujian *convergence time* yang telah dilakukan dengan cara mengirimkan paket ICMP yang dilakukan sebanyak 10 kali percobaan pada setiap algoritme, sistem telah berhasil melakukan pencarian jalur terpendek dengan *cost* paling kecil. Perhitungan *convergence time* didapatkan berdasarkan waktu yang dibutuhkan oleh *routing engine* dalam membentuk rute yang ada pada *data plane*.

Dari hasil pengujian yang telah dilakukan untuk setiap variasi jumlah *switch* pada setiap topologi akan mengalami perbedaan *convergence time*, hal ini dikarenakan apabila dilihat dari hasil pengujian, *convergence time* bertambah seiring dengan penambahan *switch*. Dapat disimpulkan pada pengujian *convergence time* algoritme *Dijkstra* lebih unggul dari algoritme *Bellman-Ford* dan *Floyd-Warshall*. Algoritme *Dijkstra* lebih unggul dalam pengujian *convergence time* selanjutnya ialah algoritme *Floyd-Warshall*, kemudian disusul dengan *Bellma-Ford*. Hal ini dikarenakan jumlah *looping* pada masing-masing algortime berbeda. Algoritme dijkstra merupakan *link state* yang tidak rentan terhadap *looping,* sehingga pencarian jalur pada algoritme *dijstra* lebih cepat Sedangkan algoritme *Floyd-warshall* membandingkan simpul dengan semua pasangan simpul yang ada. Sementara algoritme *Bellman-Ford* memiliki fitur *broadcast routing* antar tetangga untuk kemudian dibandingkan nilainya dan ditentukan jalur

terpendeknya sehingga membutuhkan waktu yang relatif lama untuk mencapai *convergance time*.

5.2.3 Analisis *Throughput*

Pengujian *Throughput* dilakukan sebanyak 5 kali percobaan dengan pengiriman paket TCP. Berdasarkan hasil pengujian yang telah dilakukan, nilai *throughput* mengalami kenaikan seiring dengan bertambahnya jumlah koneksi. Hal ini dikarenakan dengan bertambahnya jumlah koneksi maka bertambah pula jumlah pengiriman yang dilakukan untuk setiap koneksi. Dari hasil pengujian algoritme *Dijkstra* memiliki *throughput* lebih tinggi dari algoritme lain. Namun dari pengujian *throughput* yang telah dilakukan, ketiga algoritme tidak memiliki perbedaan *throughput* yang besar, namun algoritme *Floyd-Warshall* memiliki *trhoughput* paling rendah dikarenakan waktu pembentukan jalur atau *convergence tim*e lebih lama dari algoritme yang lain. Pada grafik hasil pengujian *throughput* menunjukkan bahwa, algoritme *Dijkstra, Bellman-Ford* dan *Floyd-Warshall* tidak memiliki perbedaan yang signifikan, hal ini dikarenakan jalur yang dilewati untuk mengirimkan paket adalah sama.

5.2.4 Analisis *Recovery Time* (*Link Failure*)

Recovery time merupakan waktu yang diperlukan oleh *controller* untuk membentuk jalur baru ketika terjadi *link failure* pada salah satu atau beberapa *link* topologi. Pengujian *recovery time* dilakukan sebanyak 10 kali percobaan. Dari hasil pengujian yang telah di dapat diketahui semakin banyak *switch* pada suatu topologi maka akan membuat *controller* membutuhkan *recovery time* lebih lama untuk menemukan jalur tependek yang baru. Berbeda dengan hasil pengujian *convergence time* yang menghasilkan algoritme *Dijkstra* lebih unggul, namun pada pengujian *recovery time* algoritme *Floyd-Warshall* lebih unggul karena memiliki *recovery time* yang lebih cepat dibandingkan dengan *Dijkstra* dan *Bellman-Ford*. Hal ini dikarenakan setelah pemutusan *link*, algoritme *Floyd-Warshall* masih menyimpan informasi jalur-jalur yang lain. Informasi tersebut diperoleh ketika pencarian jalur sebelumnya.

5.2.5 Analisis *Packet Loss*

Pengujian packet loss merupakan pengujian yang dilakukan untuk mengetahui seberapa besar paket yang hilang ketika pengiriman dilakukan. Berdasarkan hasil pengujian *packet loss* yang dilakukan sebanyak 5 kali menunjukkan bahwa penambahan jumlah *client* akan mempengaruhi nilai rata-rata *packet loss*. Hal ini dikarenakan semakin banyak jumlah *client* yang mengirimkan paket maka sumberdaya akan dibagikan pada jaringan tersebut semakin sedikit sehingga paket UDP yang dikirimkan akan banyak yang hilang. Dari grafik pengujian nilai *packet loss* pada algoritme *Bellman-Ford* mengalami kenaikan paling tinggi kemudian disusul oleh algoritme *Dijkstra* dan *Floyd-warshall*. Peningkatan nilai rata-rata *packet loss* cukup tinggi pada penambahan 60 *client*, karena paket UDP yang dikirimkan tidak secara maksimal menggunakan *bandwidth* yang tersedia.

Ketika pengiriman paket UDP dengan jumlah *client* yang besar maka *packet loss* juga tinggi yang dikarenakan ada paket yang *retransmission time out*.

BAB 6 PENUTUP

6.1 Kesimpulan

Berdasarkan hasil pengujian dan analisis yang telah dilakukan, dapat ditarik kesimpulan dari penelitian ini, yaitu sebagai berikut:

1. Penulis telah berhasil melakukan penelitian tentang analisis kinerja algoritme *Dijkstra, Bellman-Ford,* dan *Floyd-Warshall* pada arsitektur jaringan *Software Defined Network* untuk penentuan jalur terpendek dengan hasil cost paling kecil. Penelitian dilakukan dengan menggunakan emulator Mininet, *controller* Ryu dan topologi *Abilene*. Berikut merupakan hasil kinerja masing-masing algorime pada tiap pengujian yang dilakukan:
   a. Berdasarkan hasil pengujian validasi yang dilakukan dengan menghitung secara manual sesuai dengan pencarian jalur masing-masing algoritme, sistem telah mampu melakukan pencarian jalur terhadap yang ada dalam suatu topologi. Jalur yang dihasilkan pada sistem dan penghitungan manual adalah sama yaitu 9-10-11-6-5-4-2-1.
   b. Berdasarkan hasil pengujian *convergence time* algorime D*ijkstra* memiliki *convergence time* paling baik yaitu dengan rata-rata 0,006942ms-0,011086ms dengan jumlah *switch* 4, 6, 8 dan 11 dibandingkan dengan algorime *Bellman-Ford* yang memiliki rata-rata waktu 0,00698ms-0,013088ms dan *Floyd-Warshall* memiliki rata-rata 0,007514ms-0,04037ms.
   c. Hasil pengujian *throughput*, rata-rata *throughput* pada algoritme *Dijkstra* sekitar 1139,1-1150,8Mbps, algoritme *Bellman-Ford* 1139,8-1150,2Mbps, dan *Floyd-Warshall* memiliki *throughpu*t 1140-148,4Mbps.
   d. Berdasarkan hasil dari pengujian *recovery time*, algoritme *Dijkstra* memiliki waktu sekitar 2,99-31,51ms, algoritme *Bellman-Ford* 2,34-36,57ms, dan algoritme *Floyd-Warshall* 2,38-29,81ms.
   e. Berdasarkan hasil pengujian *packet loss*, rata-rata *packet loss* algoritma *Dijkstra* sekitar 129-411,2kb, algoritme *Bellman-Ford* 146-473,2kb dan algoritme *Floyd-Warshall* 128-433,4kb.
2. Berdasarkan hasil dari pengujian yang telah dilakukan, maka analisa dari
   a. Berdasarkan hasil pengujian validasi setiap algoritme menentukan jalur yang sama dari perhitungan manual maupun pada sistem, jalur yang dihasilkan yaitu 9-10-11-6-5-4-2-1.
   b. Berdasarkan hasil pengujian *convergence time*, *convergence time* meningkat seiring bertambahnya jumlah *switch.* Dengan hasil algoritme Dijkstra lebih unggul karena memiliki waktu lebih cepat dibandingkan algoritme *Bellman-Ford* dan *Floyd-Warshall. Floyd-warshall* memiliki *convergence time* paling buruk yang membutuhkan waktu lama untuk membentuk jalur pengiriman paket.

c. Berdasarkan hasil pengujian *throughput* untuk mengukur kualitas jaringan pada algoritme *Dijkstra* memiliki jumlah throughput lebih baik dari algoritme *Bellman-Ford* dan *Floyd-warshall.*

d. Berdasarkan hasil pengujian *recovery time* algoritme *Floyd-Warshall* memiliki waktu paling baik atau paling cepat dibandingkan dengan algoritme *Dijkstra* dan *Bellman-Ford*. Hal ini dikarenakan *Floyd-Warshall* masih memiliki jalur lain untuk pengiriman jalur.

e. Berdasarkan hasil pengujian *packet loss* algoritme *Dijkstra* memiliki jumlah *packet loss* paling rendah dibandingkan algoritme *Bellman-Ford* dan *Floyd-Warshall.*

6.2 Saran

Beberapa saran berdasarkan hasil penelitian ini untuk pengembangan penelitian selanjutnya, yaitu sebagai berikut:

1. Perlu dilakukan penambahan jumlah *switch* untuk mengetahui kinerja algoritme pada topologi yang lebih kompleks
2. Perlu dilakukan implementasi pada *real network*.

DAFTAR PUSTAKA

Nadeau & Gray, “SDN- Software Defined Network”, 2013

Mulyana Eueung, “Buku Komunitas SDN-RG”,Published with Gitbook,2015

OpenNetworking Foundation, 2012. Software Defined Networking: The New Norm for Networks.

Riza Abu.; dkk., 2016 “Analisis Performansi Perutingan Link State Menggunakan Algoritma Djikstra pada Platform Software Defined Network (SDN)”, Fakultas Teknik Elektro, Universitas Telkom Bandung

Munir, Rinaldi. 2005. “Strategi Algoritmik”.Bandung. Institut Teknologi Bandung.

Kartadie, Rikie.; Satya, Barka., Februari 2015, “Uji Performa Controller Floodlight dan OpenDayLight sebagai Komponen Utama Arsitektur Software Defined Network”, STMIK AMIKOM Yogyakarta.

G. Patel, A.S. Athreya, and S. Erukulla., “OpenFlow based Dynamic Load Balanced Switching, “ COEN 233, Project Report, 2013

Muttaqin Ahmad Rizal,”*Implementasi Network Slicing dengan Menggunakan FlowFisor untuk Mengontrol Traffic Data Packet pada Jaringan Software Defined Network”,* Fakultas Ilmu Komputer, universitas Brawijaya, 2017

Kurniawati Ulfa, “Analisis Perbandingan Algoritma Djikstra dan Bellman-Ford dengan menggunakan Software Defined Network”, Fakultas Ilmu Komputer,Universitas Brawijaya,2016

Hadiansyah Doni, *“Implementasi Penentuan Bobot Link Menggunakan Logika Fuzzy untuk Pencarian Jalur Terpendek pada Software Defined Networking*”, Fakultas Ilmu Komputer, Universitas Brawijaya,2017

Ibrahim,2016 “Analisis Perbandingan Algoritma Floyd-Warshall dan Djikstra untuk Menentukan Jalur Terpendek pada jaringan OpenFLow”, Fakultas Ilmu Komputer, Universitas Brawijaya

Stanford, University IT, 2015

Gopinda Al Araaf, Mei 2016 **”**Implementasi Algoritma Bellman-Ford dan Floyd Warshall untuk Mencari Rute Terpendek (Stud Kasus: Rute Jakarta-Jogja)”, Sistem Informasi STMIK AMIKOM Yogyakarta,

Lazuardi Erico, 2016 “Metode Pemilihan Jalur Routing Adaptif Berdasarkan Kemacetan Jaringan Dengan Algoritma Dijkstra pada OpenFlow”, Fakultas Ilmu Komputer, Universitas Brawijaya

NetworkX Developers (2015). *NetworkX*. Dipetik September 7, 2017, dari http://networkx.github.io/

Anam Khoerul, "*Analisis Performa Jaringan Software Defined Network Berdasarkan penggunaan Cost pada Protocol Routing Open Shortest Path First*", Universitas Gajah Mada, 2017

Ramadhan Faizal,"*Implementasi Routing Berbasis Algoritme Dijkstra Pada Software Defined Networking Menggunakan Kontroler Open Network Operation System*",Fakultas Ilmu Komputer, Universitas Brawijaya, 2017

Manullang Romy Dwi Andika,"*Implementasi K-Shortest Path Routing pada Jaringan Software Defined Network"* Fakultas Ilmu Komputer, Universitas Brawijaya, 2017