# ANALISIS PERBANDINGAN PERFORMANSI ALGORITME FLOYD-WARSHALL DAN ALGORITME JOHNSON UNTUK PENENTUAN RUTE TERPENDEK PADA *SOFTWARE DEFINED NETWORK*

**SKRIPSI**

**KEMINATAN TEKNIK KOMPUTER**

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Mohamad Ilham Firdaus
NIM: 135150300111038


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

# PENGESAHAN

Analisis Perbandingan Performansi Algoritme Floyd-Warshall Dan Algoritme Johnson Untuk Penentuan Rute Terpendek Pada *Software Defined Network*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Mohamad Ilham Firdaus
NIM: 135150300111038

Skripsi ini telah diuji dan dinyatakan lulus pada
12 Januari 2018
Telah diperiksa dan disetujui oleh:

| Dosen Pembimbing I | Dosen Pembimbing II |
|---|---|
| Rakhmadhany Primananda, S.T, M.Kom | Mochammad Hannats Hanafi Ichsan, S.ST, M.T |
| NIK. 201609 860406 1 001 | NIK. 201405 881229 1 001 |

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D.
NIP. 19710518 200312 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 didalam 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, 12 Januari 2018

Mohamad Ilham Firdaus
NIM. 135150300111038

# KATA PENGANTAR

Puji syukur kehadirat Allah SWT yang telah memeberikan rahmat, taufik dan hidayah-Nya sehingga penulis dapat menyelesaikan laporan skripsi yang berjudul "Analisis Perbandingan Performansi Algoritme Floyd-Warshall Dan Algoritme Johnson Untuk Penentuan Rute Terpendek Pada *Software Defined Network*".

Banyak kesulitan dan hambatan yang dialami penulis dalam penyusunan skripsi ini, tetapi semua itu dapat diatasi berkat dukungan dan bantuan dari berbagai pihak. Oleh karena itu penulis ingin menyampaikan rasa hormat dan terima kasih kepada:

1. Allah SWT dan Nabi Muhammad SAW karena atas kehendak, rahmat dan nikmat-Nya laporan skripsi ini telah selesai dengan baik.
2. Kedua orang tua dan seluruh keluarga besar atas segala nasehat, kasih sayang, perhatian, dan kesabarannya memberikan semangat kepada peneliti, serta senantiasa tiada hentinya memberikan doa demi terselesaikannya skripsi ini.
3. Bapak Tri Astoto Kurniawan, S.T, M.T, Ph.D. selaku Ketua Jurusan Teknik Informatika Universitas Brawijaya Malang.
4. Bapak Sabriansyah Rizqika Akbar, S.T., M.Eng selaku Ketua Program Studi Teknik Komputer Universitas Brawijaya Malang.
5. Bapak Rakhmadhany Primananda, S.T, M.Kom selaku dosen pembimbing pertama yang telah memberikan pengarahan dan bimbingan kepada penulis sehingga dapat menyelesaikan skripsi ini.
6. Bapak Mochammad Hannats Hanafi Ichsan, S.ST, M.T selaku dosen pembimbing kedua yang telah memberikan pengarahan dan bimbingannya kepada penulis sehingga dapat menyelesaikan skripsi ini.
7. Seluruh civitas akademika Informatika Universitas Brawijaya dan terkhusus untuk teman-teman Teknik Komputer Angkatan 2013 yang telah banyak memberi bantuan dan dukungan selama peneliti menempuh studi di Teknik Komputer Universitas Brawijaya dan selama penyelesaian skripsi ini.
8. Tim Lulus Bareng yang telah memberikan banyak motivasi selama penulis menempuh studi dan menyelesaikan skripsi.
9. Keluarga Warriors Aksa 7 khususnya Warriors Aksa 7 Regional Malang yang telah memberikan banyak motivasi selama penulis menempuh studi dan menyelesaikan skripsi.
10. Arek-arek Kontrakan Ganteng yang telah memberikan banyak motivasi selama penulis menempuh studi dan menyelesaikan skripsi.
11. Seluruh pihak yang tidak dapat diucapkan satu persatu, peneliti mengucapkan banyak terima kasih atas segala bentuk dukungan dan doa sehingga laporan skripsi ini dapat terselesaikan.

Penulis menyadari bahwa skripsi ini masih banyak kekurangan, oleh karena itu untuk segala kritik dan saran yang membangun penulis ucapkan terima kasih.

Malang, 12 Januari 2018

Penulis

ilhamfirdaa@gmail.com

# ABSTRAK

Teknologi jaringan komputer yang terus berkembang pesat berdampak pada kebutuhan yang tinggi terhadap kinerja dan kontrol jaringan yang semakin kompleks. Konfigurasi jaringan, jumlah permintaan dan metode manajemen jaringan merupakan faktor yang dapat mempengaruhi kinerja suatu jaringan. Salah satu yang mempengaruhi manajemen jaringan tersebut adalah *protocol routing*. Pada jaringan konvensional, konfigurasi *protocol routing* tidak fleksibel, tidak efisien dan perlu mengkonfigurasi setiap perangkat jaringan, terlebih jika menggabungkan beberapa vendor yang berbeda dalam satu jaringan. Salah satu solusi untuk permasalahan tersebut adalah SDN (*Software Defined Network*). SDN adalah model arsitektur yang menawarkan konsep yang memisahkan antara *control plane* dengan *data plane*. *Control plane* SDN yang bersifat programmable memungkinkan untuk menerapkan berbagai aplikasi jaringan, salah satunya *routing*. Algoritme *routing* yang digunakan pada penelitian ini adalah Algoritme Floyd-Warshall dan Algoritme Johnson. Kedua algoritme akan diuji coba, dianalisis dan dibandingkan kinerjanya. Pengujian dilakukan dengan mengamati beberapa parameter, antara lain *packet loss*, *delay*, *convergence time*, CPU dan *memory usage*. Kedua algoritme diimplementasikan pada *controller* ryu dan menggunakan mininet sebagai simulator jaringan SDN. Pada pengujian *packet loss* tidak terjadi perbedaan yang signifikan, Floyd-Warshall memiliki rata-rata 2,33% berbanding Johnson dengan rata-rata 2,48%. Begitu pun pada pengujian *delay*, rata-rata *delay* Floyd-Warshall 4,08 ms berbanding dengan rata-rata Johnson 4,02 ms. Pada hasil pengujian *convergence time* Johnson lebih cepat saat jumlah *switch* 6 dan 10, sedangkan Floyd-Warshall lebih unggul saat jumlah *switch* 14. Pada pengujian CPU usage Johnson mengonsumsi resource lebih besar dengan rata-rata 23,84% dibandingkan Floyd-Warshall dengan rata-rata 21,78%. Sedangkan untuk hasil *memory usage* kedua algoritme sama-sama mengonsumsi *memory* sebesar 1,3%.

**Kata kunci:** SDN, Ryu, Mininet, Floyd-Warshall, Johnson

# *ABSTRACT*

*The growth of computer network technology has an impact on the high demand for network performance and increasingly control network complexity. Network configuration, requests and network management method are factors that affect the performance of a network. One that affect network management is routing protocol. In a conventional network, routing protocol configuration is inflexible, inefficient and needs to configure each network device, especially if it combine several different vendors in a network. One of the solution for this problem is SDN (software defined network). SDN is an architecture model that offer a concept to separate control plane and data plane. The programmable SDN control plane make it possible to implement network applications, one of which is routing. This research uses two kind of shortest path algorithm those are Floyd-Warshall algorithm and Johnson algorithm, the performance of these algorithms will be tested, analyzed, and compared. Some paramaters are observing such as packet loss, delay, convergence time, CPU and memory usage. Both algorithms are implemented in ryu controller and mininet as SDN network simulator. Packet loss testing has no significant difference, Floyd-Warshall has an average of 2,33% compared to Johnson with an average of 2,48%. Similarly in the delay testing, the average delay of Floyd-Warshall is 4,08 ms compared to Johnson with an average of 4,02 ms. Convergence time testing, Johnson is faster when the number of switches 6 and 10, while Floyd-Warshall its better when the number of switches 14. In CPU usage testing, Johnson consumes more resources with an average of 23,84% compared to Floyd-Warshall with an average of 21,78%. While the result of memory usage of both algorithm equally consume 1,3% of memory.*

***Keywords:*** *SDN, Ryu, Mininet, Floyd-Warshall, Johnson*

# DAFTAR ISI

# DAFTAR TABEL

# DAFTAR GAMBAR

# BAB 1 PENDAHULUAN

## 1.1 Latar Belakang

Protokol kontrol terdistribusi dan *transport network* berjalan pada *router* dan *switch* yang merupakan komponen utama untuk pertukaran informasi dalam bentuk paket digital. Pada jaringan konvensional, konfigurasi *protocol routing* tidak fleksibel, tidak efisien dan perlu mengkonfigurasi setiap perangkat jaringan, terlebih lagi jika menggabungkan beberapa vendor yang berbeda dalam satu jaringan. Selain kompleksitas konfigurasi, lingkungan jaringan harus menyesuaikan diri terhadap perubahan jaringan yang terjadi. Mekanisme rekonfigurasi dan respon otomatis hampir tidak ada dalam jaringan konvensional. Hal ini tentu tidak dapat memenuhi tuntutan operasional saat ini yang memiliki jaringan besar dan perangkat jaringan yang memiliki spesifikasi berbeda (Kreutz, et al., 2014).

*Software Defined Network* (SDN) merupakan paradigma baru arsitektur jaringan yang menawarkan konsep *control plane* yang dipisahkan dari *data plane*. Pemisahan ini mendefinisikan *control plane* akan diletakkan secara terpusat pada sebuah *controller* yang bertugas untuk mengontrol perilaku suatu jaringan komputer dan perangkat jaringan (*switch* atau *router*) yang berada pada *data plane* menjadi perangkat untuk *forwarding* paket data. Pemisahan *control plane* dan *data plane* dapat direalisasikan dengan menggunakan *application programming interface* (API). *Network administrator* menggunakan API tersebut untuk mengotomatisasi, mengatur dan mengoperasikan jaringan. Sehingga tidak perlu mengetahui perbedaan konfigurasi *syntax* atau *semantic* pada jaringan yang berbeda, sebab API pada *controller* dapat berkomunikasi pada perangkat dari vendor yang berbeda (Nadeau & Gray, 2013)

Perkembangan teknologi jaringan komputer yang semakin maju berdampak pada kebutuhan yang tinggi terhadap kinerja jaringan dan kontrol jaringan yang semakin kompleks. Konfigurasi jaringan, jumlah permintaan dan metode manajemen jaringan merupakan faktor yang dapat mempengaruhi performansi suatu jaringan. Salah satu yang mempengaruhi manajemen jaringan tersebut adalah *routing*. *Routing* dibutuhkan untuk mendapatkan rute dari satu jaringan ke jaringan lainnya. Dengan sifatnya yang *programmable*, *controller* SDN memungkinkan untuk mengimplementasikan protokol *routing*. Untuk membangun protokol *routing* yang baik diperlukan algoritme *routing* yang dapat menentukan rute terpendek (*shortest path*) pada berbagai macam topologi tanpa melakukan konfigurasi ulang (Kurose & Ross, 2013).

Pada penelitian ini menggunakan Algoritme Floyd-Warshall dan Algoritme Johnson. Baik Algoritme Floyd-Warshall maupun Algoritme Johnson keduanya bersifat *all-pair shortest path.* Berbeda dengan Dijkstra atau Bellman-Ford yang berfsifat *single source shortest path*, algoritme *routing all-pair shortest path* akan mencari rute terpendek berdasarkan semua pasangan *node*, penentuan rute terpendek cepat dan performansinya stabil (Saputra, 2016). Floyd-Warshall dan Johnson dipilih karena sama-sama bersifat *all-pair shortest path* dan juga karena

memiliki karateristik yang bertolak-belakang dimana Floyd-Warshall yang efektif pada topologi dengan *node* padat/banyak, sedangkan Johnson efektif pada topologi dengan *node* yang renggang/sedikit.

Pada penelitian sebelumnya oleh Ihsan Aris Saputra dengan judul "Uji Performansi Algoritme Floyd-Warshall pada Jaringan *Software Defined Network*" didapatkan hasil waktu *link failure* yang lebih cepat dari *convergence time*, *overhead traffic* berbanding lurus dengan penambahan *switch* dan hasil QoS (*delay* dan *packet loss*) masih dalam rentang nilai yang ditetapkan pada ITU-T G.101 (Saputra, 2016). Sedangkan pada penelitian yang dilakukan oleh Muhammad Ilhamsyah dengan judul "Simulasi dan Uji Kinerja Algoritme Johnson Untuk Penentuan Rute Terbaik pada Jaringan *Software Defined Network*" didapatkan hasil *convergence time* yang menunjukkan setiap terjadi penambahan *switch* maka terjadi peningkatan waktu serta hasil dari *link failure* yang menunjukkan bahwa *routing* yang dirancang bersifat fleksibel karena dapat menyesuaikan bila terjadi perubahan pada jaringan (Ilhamsyah, 2016).

Oleh karena itu, penelitian ini bertujuan untuk membandingkan kinerja Algoritme Floyd-Warshall dan Algoritme Johnson dalam menentukan rute terpendek dalam pemilihan jalur pada SDN. Parameter yang digunakan untuk mengukur kinerja algoritme diantaranya adalah *delay*, *packet loss*, *convergence time*, CPU dan *memory usage*. Diharapkan penelitian ini mampu memberikan alternatif pemilihan algoritme *routing* pada SDN.

## 1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dikemukakan, rumusan masalah pada penelitian ini antara lain:

1. Bagaimana perancangan dan implementasi Algoritme Floyd-Warshall dan Algoritme Johnson pada *software defined network*?
2. Bagaimana pengujian yang dilakukan pada *software defined network* menggunakan Algoritme Floyd-Warshall dan Algoritme Johnson?
3. Bagaimana hasil analisis perbandingan Algoritme Floyd-Warshall dan Algoritme Johnson?

## 1.3 Tujuan

Tujuan yang ingin dicapai pada penelitian ini adalah:

1. Dapat merancang dan mengimplementasikan Algoritme Floyd-Warshall dan Algoritme Johnson pada *software defined network.*
2. Dapat melakukan pengujian Algoritme Floyd-Warshall dan Algoritme Johnson pada *software defined network* menggunakan parameter yang telah ditentukan.
3. Mengetahui hasil analisis perbandingan performansi Algoritme Floyd-Warshall dan Algoritme Johnson pada *software defined network.*

## 1.4 Manfaat

Memberikan gambaran implementasi protokol *routing* menggunakan Algoritme Floyd-Warshall dan Algoritme Johnson dan menganalisa serta membandingkan kinerjanya dalam penentuan rute terpendek sehingga mampu memberikan alternatif penentuan algoritme *routing* pada SDN.

## 1.5 Batasan Masalah

Agar pembahasan dalam penelitian ini dapat dilakukan secara terarah dan mendapatkan hasil sesuai dengan yang diharapkan, maka perlu diterapkan batasan permasalahan pada sistem yang akan dibuat. Batasan masalah yang akan dibahas adalah sebagai berikut:

1. Implementasi sistem berupa simulasi.
2. *Controller* yang digunakan adalah *ryu*.
3. Bahasa pemrograman yang digunakan adalah *python*.
4. Simulator yang digunakan adalah mininet.
5. Topologi menggunakan 10 *switch* dan 6 host.

## 1.6 Sistematika Pembahasan

Sistematika pembahasan skripsi yang disusun ini akan dibahas pada bab-bab yang akan diuraikan di bawah ini:

**BAB I Pendahuluan**

Bab ini menguraikan latar belakang, merumuskan inti permasalahan, menentukan tujuan yang ingin dicapai, manfaat yang dapat diperoleh, batasan masalah, dan sistematika penulisan dari penelitian yang dilakukan yaitu "Analisis Perbandingan Performansi Algoritme Floyd-Warshall Dan Algoritme Johnson Untuk Penentuan Rute Terpendek Pada *Software Defined Network*".

**BAB II Landasan Kepustakaan**

Bab ini mencakup kajian tentang penelitian sebelumnya yang relevan dan menjelaskan landasan teori yang terkait dengan penelitian yang didapat dari berbagai referensi yang relevan untuk digunakan sebagai panduan dalam penelitian.

**BAB III Metodologi Penelitian**

Bab ini membahas berbagai alur kerja yang digunakan dalam penelitian yang dilakukan. Langkah kerja tersebut adalah studi literatur, analisis kebutuhan sistem yang terdiri dari kebutuhan fungsional dan kebutuhan non-fungsional, perancangan sistem yang dijelaskan dengan flowchart dan implementasi Algoritme Floyd-Warshall dan Johnson dalam bentuk simulasi.

**BAB V Perancangan dan Implementasi**

Bab ini menjelaskan perancangan dan implementasi sistem yang meliputi instalasi perangkat lunak, perancangan topologi dan perancangan algoritme *routing* yaitu Algoritme Floyd-Warshall dan Algoritme Johnson.

**BAB VI Pengujian dan Analisis**

Bab ini berisi pengujian, analisis dan perbandingan hasil dari subbab sebelumnya. Perbandingan antara kedua algoritme dengan melihat beberapa parameter pengujian yang telah ditentukan. Parameter pengujian diantaranya adalah *delay*, *packet loss*, *convergence time*, CPU dan *memory usage*.

**BAB VII Penutup**

Bab ini mengemukakan kesimpulan yang diperoleh dari hasil penelitian dan saran yang berisi hal-hal yang perlu dilakukan untuk pengembangan selanjutnya, agar dapat dilakukan perbaikan di masa yang akan datang.

# BAB 2 LANDASAN KEPUSTAKAAN

Pada penelitian ini, penulis mengkaji dari buku pedoman tentang konsep *software defined network* yang berjudul "SDN: *Software Defined Network*" ditulis oleh Thomas D. Nadeau & Ken Gray. Adapun jurnal penelitian sebelumnya yang penulis jadikan referensi mengenai pengujian Algoritme Floyd-Warshall pada SDN yang berjudul "Uji Performansi Algoritme Floyd-Warshall pada Jaringan *Software Defined Network*" ditulis oleh Ihsan Aris Saputra. Jurnal yang dikaji berkaitan dengan penerapan Algoritme Johnson yang berjudul "Simulasi dan Uji Kinerja Algoritme Johnson untuk Penentuan Rute Terbaik pada Jaringan *Software Defined Network*" ditulis oleh Muhammad Ilhamsyah. Jurnal lain yang membahas penerapan algoritme lainnya pada SDN berjudul "Analisis Perbandingan Algoritme Dijkstra dan Bellman-Ford untuk Menentukan Rute Terpendek dengan Menggunakan Jaringan *Software Defined Network*" ditulis oleh Ulfa Kurniawati. Sedangkan untuk beberapa referensi dasar teori sebagai pengetahuan tentang teknologi yang digunakan meliputi protokol openflow, *controller ryu*, simulator mininet dan topologi.

## 2.1 Tinjauan Pustaka

Tinjauan pustaka akan berisi tentang perbandingan antara penelitian yang pernah dilakukan dengan rencana penelitian yang akan dilakukan, berikut adalah perbandingannya:

**Tabel 2.1 Kajian Pustaka**

| No | Nama Penulis, Tahun dan Judul | Persamaan | Perbedaan | |
|---|---|---|---|---|
| | | | Penelitian Terdahulu | Rencana Penelitian |
| 1. | Ihsan Aris Saputra [2016] Uji Performansi Algoritme Floyd-Warshall pada Jaringan *Software Defined Network.* Universitas Telkom | Implementasi Algoritme Floyd-Warshall pada *software defined network* | Pengujian performansi Algoritme Floyd-Warshall terhadap *convergence time*, overhead traffic, *delay* dan *packet loss* | Membandingkan performansi Algoritme Floyd-Warshall dan Johnson terhadap parameter packel loss, *delay*, *convergence time*, CPU dan *memory usage* |
| 2. | Muhammad Ilhamsyah [2016] Simulasi dan Uji Kinerja Algoritme | Implementasi Algoritme Johnson pada | Pengujian kinerja Algoritme Johnson | Membandingkan performansi Algoritme Floyd-Warshall dan |

| | | | | |
|---|---|---|---|---|
| | Johnson untuk Penentuan Rute Terbaik pada Jaringan *Software Defined Network.* Universitas Telkom | *software defined network* | terhadap *network* convergence, overhead traffic, resource utilization dan *link* failure | Johnson terhadap parameter packel loss, *delay*, *convergence time*, CPU dan *memory usage* |
| 3. | Ulfa Kurniawati [2016] Analisis Perbandingan Algoritme Dijkstra dan Bellman-Ford untuk Menentukan Rute Terpendek dengan Menggunakan Jaringan *Software Defined Network.* Universitas Brawijaya | Implementasi Algoritme Shortest Path pada *software defined network* | Menggunakan Algoritme Dijkstra dan Bellman-Ford | Menggunakan Algoritme Floyd-Warshall dan Johnson |

## 2.2 Dasar Teori

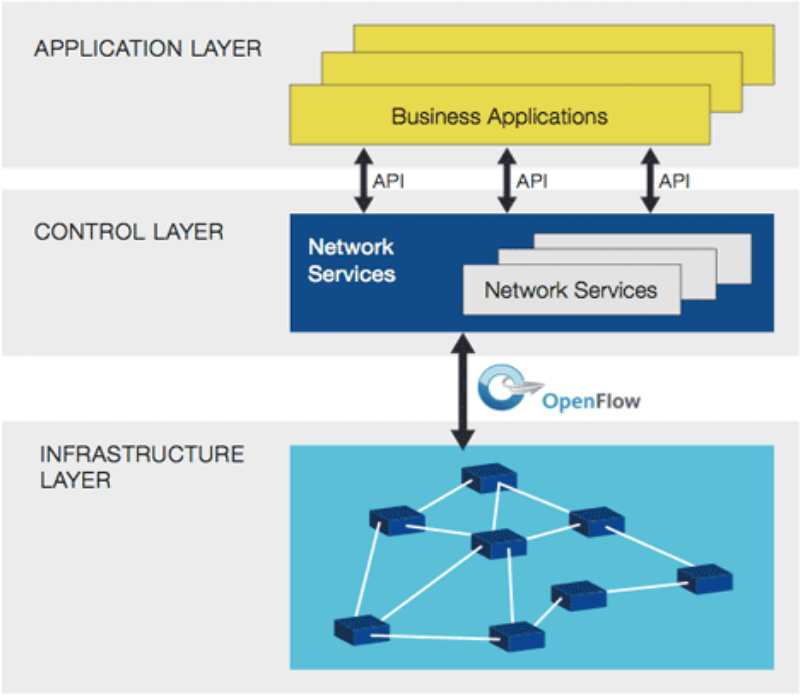
Berdasarkan beberapa informasi yang didapat dari tinjauan pustaka, maka dalam penelitian ini terdapat beberapa dasar teori antara lain:

### 2.2.1 *Software Defined Network*

*Software Defined Network* (SDN) merupakan arsitektur jaringan terkini yang sedang dikembangkan secara pesat beberapa tahun ini. SDN mengusung konsep arsitektur jaringan yang memisahkan antara *control plane* dengan *data plane*, dimana *control plane* akan diletakkan secara terpusat pada sebuah *controller*. Pemisahan *control plane* dan *data plane* dapat direalisasikan dengan menggunakan *application programming interface* (API). *Network administrator* menggunakan API tersebut untuk mengotomatisasi, mengatur dan mengoperasikan jaringan. Sehingga tidak perlu mengetahui perbedaan konfigurasi *syntax* atau *semantic* pada jaringan yang berbeda, sebab API pada *controller* dapat berkomunikasi pada perangkat dari vendor yang berbeda (Nadeau & Gray, 2013).

Arsitektur SDN tersusun dari tiga layer yang terdiri dari *application layer*, *control layer* dan *infrastructure layer*. *Application layer* berada pada lapisan paling atas, layer ini merupakan letak aplikasi SDN yang diprogram untuk berkomunikasi dengan *controller* SDN menggunakan *application programming interface* (API). Pada *control layer* terdapat SDN *controller* yang berfungsi menerima intruksi dari SDN *application* dan menyalurkannya pada perangkat jaringan yang dijembatani oleh protokol openflow. *Controller* juga mengekstraksi informasi jaringan dari

perangkat keras dan berkomunikasi balik ke SDN *application*. Lapisan paling bawah yaitu *infrastructure layer* adalah perangkat jaringan yang berfungsi untuk *forwarding* dan *processing data path* (Open Networking Foundation, n.d.). Untuk lebih jelasnya, Gambar 2.1 merupakan gambar arsitektur SDN.



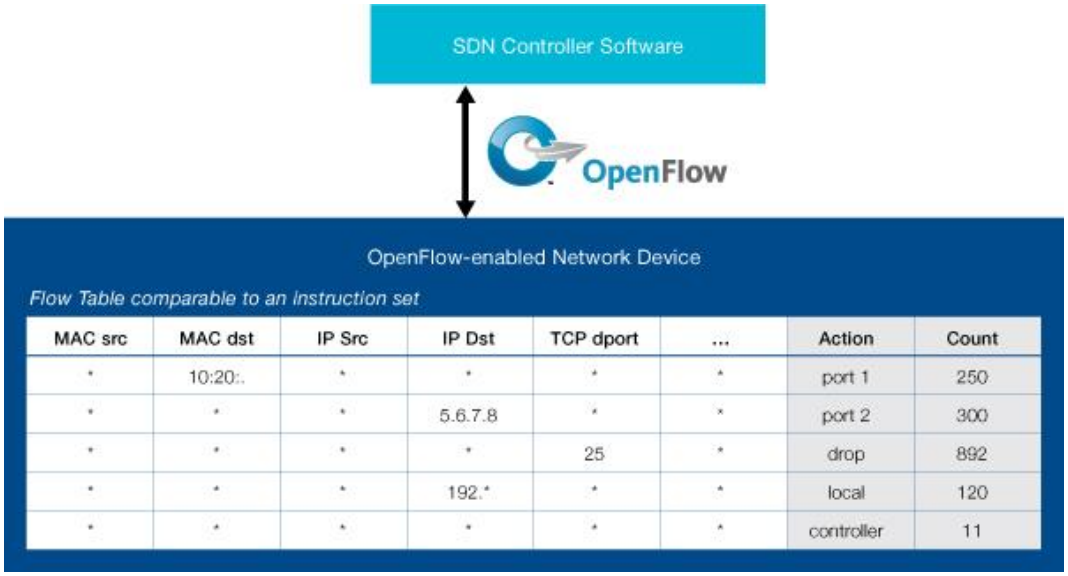

**Gambar 2.1 Arsitektur SDN**

**Sumber**: (www.open*network*ing.org)

### 2.2.2 OpenFlow

Pada SDN, *network intelligence* terpusat pada *software* berbasis *controller* (*control plane*) dan *network device* menjadi perangkat yang berfungsi untuk meneruskan paket (*data plane*) yang dapat diprogram melalui *open interface*. Salah satu implementasi dari open interface ini disebut openflow. *Openflow* memungkinkan akses langsung untuk memanipulasi *forwarding plane* pada perangkat jaringan seperti *switch* dan *router* (Azodolmolky, 2013).

Protokol openflow melakukan mekanisme komunikasi dengan *switch* menggunakan openflow *channel* yang terdapat pada openflow *switch*. Protokol openflow melakukan sentralisasi terhadap kerumitan jaringan ke dalam sebuah *controller software*, sehingga administrator dapat melakukan pengaturan jaringan melalui *controller* tersebut dengan mudah. *Openflow* memiliki struktur cara kerja sebagai berikut:




**Gambar 2.2 Flow *table* pada** openflow

**Sumber:** (www.open*network*ing.org)

Berdasarkan Gambar 2.2 dapat diketahui struktur *flow table* dan *action* secara rinci. *Flow table* memiliki peran untuk menentukan paket yang masuk ke dalam suatu *flow* lalu mengambil tindakan terhadap paket tersebut. *Controller* melalui openflow memperbolehkan manipulasi secara langsung seperti menambah, menghapus maupun melakukan modifikasi sebuah *flow* pada *flow tabel*. Paket *header open flow* berisikan informasi tentang source MAC, *destination* MAC, *source* IP, *destination* IP, dan TCP *port* yang akan menentukan kemana arah paket akan diteruskan. *Action* memberikan informasi terkait tindakan yang harus diambil terhadap suatu *flow*. Sedangkan counter berisi informasi jumlah paket yang diproses pada tiap *flow* (McKeown, et al., 2008).

## 2.2.3 Controller

*Controller* merupakan bagian dari SDN yang berfungsi untuk mengontrol logika pada jaringan. Seluruh kebijakan jaringan seperti *routing*, *switching* maupun pengaturan jaringan lainnya diatur oleh *controller*. Pengaturan fungsional tersebut dikembangkan melalui modul-modul pada *controller* yang dapat diprogram menggunakan bahasa pemrograman tertentu. Tiap *controller* menggunakan bahasa pemrogramannya masing-masing. SDN memiliki beberapa *controller* diantaranya NOX, POX, OpenDayLight, FloodLight, Ryu, Pyretic, dan lain-lain.

### *2.2.3.1 Ryu*

Ryu merupakan framework berbasis komponen *software defined network* yang menyediakan komponen software dengan API yang membuatnya mudah untuk mengembangkan dan mengelola aplikasi. Ryu mendukung berbagai protokol untuk mengelola perangkat jaringan, seperti OpenFLow, NetConf, OF-Config dan sebagainya (Ryu Development Team, 2017).

*Ryu* menyediakan kumpulan komponen yang berguna untuk aplikasi SDN dan dapat memodifikasi komponen yang sudah ada atau menerapkan komponen baru. Hal ini membuat programmer memungkinkan untuk menggabungkan satu komponen dengan komponen lainnya untuk membangun sebuah aplikasi SDN sesuai yang dibutuhkan.

### 2.2.4 Simulator

*Simulator* adalah sebuah aplikasi mul*time*dia yang dapat digunakan untuk menyimulasikan suatu alat tetapi tidak dapat mengganti infrastruktur aslinya. *Software defined network* memiliki beberapa simulator yang dapat mempermudah dalam menyimulasikan SDN, antara lain Mininet, EstiNet dan NS3.

#### *2.2.4.1 Mininet*

*Mininet* menciptakan jaringan virtual yang realistis, menjalankan *kernel* secara nyata, *switch* dan kode aplikasi pada satu mesin (VM, *cloud* atau *native*) (Mininet, n.d.). *Mininet* mempermudah interaksi dengan jaringan menggunakan CLI dan API, simulator ini mendukung topologi yang kompleks dan *user-defined*.

Mininet adalah sebuah simulator jaringan yang mendukung protokol openflow untuk arsitektur *software defined network*. Keunggulan mininet salah satunya mampu membangun topologi yang cukup kompleks dan sesuai keinginan perancangnya. Untuk mempermudah ketika merancang topologi, terdapat *miniedit* sebagai virtualisasinya. Sehingga tidak perlu menulis *syntax* pada *command line* pada terminal untuk membuat topologi.

### 2.2.5 *Routing*

*Routing* merupakan suatu protokol yang digunakan untuk mendapatkan rute dari satu jaringan ke jaringan yang lain. Tujuan utama dari protokol *routing* adalah untuk membangun tabel *routing*. Dimana tabel ini berisi informasi jaringan dan *interface* yang berhubungan dengan jaringan tersebut. Protokol *routing* diterapkan pada *router* untuk mengatur informasi yang diterima dari *router* lain dan *interface*nya masing-masing.

Informasi yang terdapat pada tabel *routing* dapat diperoleh secara *static routing* maupun *dynamic routing*. *Static routing* merupakan sebuah mekanisme pengisian tabel *routing* yang dilakukan dengan mengkonfigurasi secara manual pada tiap *router* oleh *network administrator*. Mekanisme *static routing* tidak cocok diterapkan pada jaringan berskala besar. Jika terdapat penambahan / perubahan topologi jaringan, maka *administrator* harus mengkonfigurasi ulang tabel *routing* pada tiap *router*. *Dynamic routing* bersifat fleksibel. Jika terjadi perubahan pada topologi atau terdapat masalah di jaringan, maka *router* akan mengetahui perubahan tersebut. D*ynamic routing* lebih responsive terhadap perubahan pada jaringan, akan tetapi lebih rentan terhadap masalah seperti *routing loops* (Kurose & Ross, 2013).

### 2.2.6 Algoritme *Routing*

Algoritme *routing* adalah dasar dari *dynamic routing* yang muncul dari perkembangan *dynamic routing*. *Algoritme routing* merupakan metode untuk menentukan rute paket pada *node*. Untuk setiap *node* pada jaringan, algoritme menentukan tabel *routing*nya.

Algoritme *routing* merupakan faktor utama yang mempengaruhi kinerja *routing* pada suatu jaringan. Tujuan dari algoritme *routing* adalah menentukan keputusan *router* untuk menentukan rute terbaik yang akan diambil dari satu jaringan ke jaringan lainnya. Suatu algoritme harus mampu mencari rute terbaik ketika terdapat jalur yang rusak (Kurose & Ross, 2013).

Perkembangan algoritme *shortest path* untuk menentukan rute terbaik saat ini menawarkan mekanisme yang efisien. Salah satunya adalah Algoritme Floyd-Warshall yang menghitung jalur terpendek dengan mencari semua jarak dari setiap *node*. Sedangkan Algoritme Johnson merupakan perpaduan dari Algoritme Dijsktra dan Algoritme Bellman-Ford. Baik Algoritme Floyd-Warshall maupun Algoritme Johnson merupakan algoritme *all-pair shortest path* yaitu algoritme *routing* yang menghitung rute terpendek dengan membandingkan setiap pasangan *node*.

#### *2.2.6.1 Algoritme Floyd-Warshall*

Algoritme Floyd-Warshall adalah salah satu varian dari dynamic programming, 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.

Algoritme Floyd-Warshall menghitung rute terpendek setiap pasangan *node* pada graf berarah. Pada sisi (*edge*) diperbolehkan memiliki bobot negatif, akan tetapi tidak diperbolehkan bagi graf untuk memiliki siklus dengan bobot negatif atau perulangan pada jalur yang diambil (Kamayudi, 2006).

Algoritme Floyd-Warshall untuk mencari rute terpendek adalah sebagai berikut:

$W = W_0$

Untuk k = 1 hingga n, lakukan:

    Untuk i = 1 hingga n, lakukan:

        Untuk j = 1 hingga n lakukan:

        Jika W[i,j] > W[i,k] + W[k,j], maka

        Tukar W[i,j] dengan W[i,k] + W[k,j]

W* = W

**Gambar 2.3 Algoritme Floyd-Warshall**

Algoritme Floyd-Warshall membentuk n matriks sesuai dengan iterasi-k untuk mencari path terpendek. Hal ini menyebabkan waktu prosesnya lambat, terutama untuk n yang besar. Meskipun waktu prosesnya bukanlah yang tercepat, Algoritme Floyd-Warhsall banyak digunakan untuk menghitung path terpendek karena kesederhanaannya.

#### *2.2.6.2 Algoritme Johnson*

Algoritme Johnson adalah perpaduan dari Algoritme Dijkstra dan Algoritme Bellman-Ford yang juga merupakan algoritme *shortest path*, akan tetapi hanya untuk satu *source* saja. Dalam segi *runtime*, menggunakan Algoritme Dijsktra untuk menentukan rute terpendek memang lebih baik daripada Alogirtma Floyd-Warshall, akan tetapi Algoritme Dijkstra tidak dapat bekerja jika terdapat bobot negatif pada *edge*.

Konsep dari Algoritme Johnson adalah dengan merubah bobot pada seluruh *edge* dan membuatnya positif menggunakan Algoritme Bellman-Ford, kemudian menghitung rute terpendek setiap pasangan *node* menggunakan Algoritme Dijkstra (Brilliant, n.d.).

Algoritme Johnson untuk mencari rute terpendek adalah sebagai berikut:

1. Merekonstruksi graf baru dengan cara menambahkan *node* baru sehingga persamaannya:

$$V' = V + (s) dan E + \{(s, v) v di V\} \quad 2.1$$

2. Setiap *node* v di V

$$0 \rightarrow w(s, v) \quad 2.2$$
$$\infty \rightarrow w(s, v) \quad 2.3$$

3. Menjalankan Algoritme Bellman-Ford pada graf baru
   a. Jika terdapat bobot negatif maka selesai
   b. Jika tidak terdapat bobot negatif maka hitung $d(s, v), \forall v \in V$
4. Setiap (u,v) di E

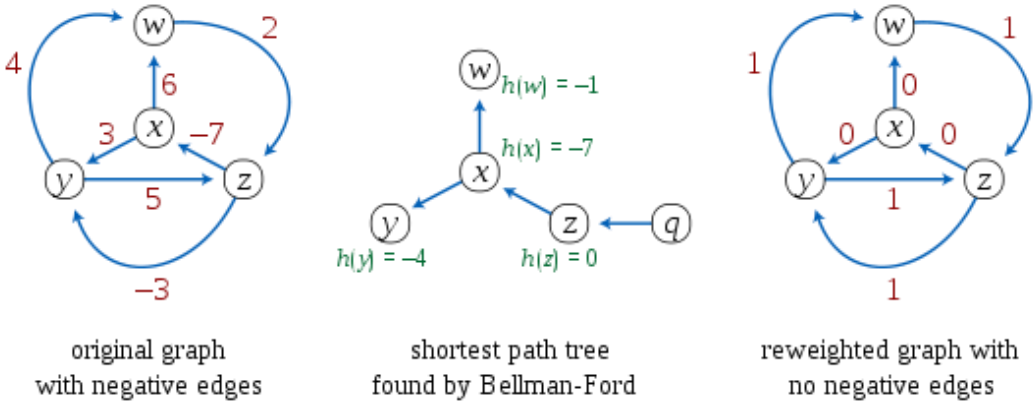
$$W(u, v) = w(u, v) + d(s, v) - d(s, u) \quad 2.4$$

5. Setiap v di V, dijalankan Algoritme Djikstra untuk menghitung d(u, v)

$$D' = d(u, v) \quad 2.5$$

6. Setiap (u, v) di V ; d(u, v) = d(u, v) + d(sv) – d(s, u)

$$D = d(u, v) \quad 2.6$$




**Gambar 2.4 Algoritme Johnson**

Berdasarkan Gambar 2.4 terdapat beberapa langkah dalam menghitung rute terpendek menggunakan Algoritme Johnson:

1. Pada gambar sebelah kiri, terdapat sebuah graf yang memiliki bobot negatif tapi bukan siklus negatif.
2. Pada gambar di tengah, ditambahkan satu *node* baru yaitu q yang dihubungkan pada seluruh *node* dengan bobot 0. Lalu Algoritme Bellman-Ford dijalankan untuk menghitung bobot minimum dengan q sebagai titik awal.
3. Pada gambar sebelah kanan, bobot pada graf telah diperbaharui. Pada graf ini semua bobot bernilai positif. Kemudian *node* baru dihilangkan dan Algoritme Dijkstra diterapkan untuk menghitung rute terpendek dari setiap *node* yang bobotnya telah diperbaharui.

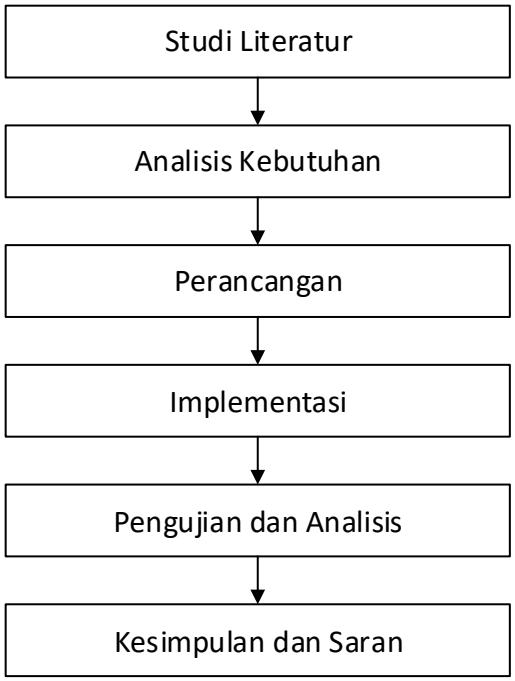
### 2.2.7 Topologi

Topologi jaringan adalah studi tentang pengaturan dan pemetaan elemen (*link*, *node*, dsb) dari interkoneksi jaringan antar *node*. Topologi dapat bersifat fisik atau logis. Topologi bersifat fisik berarti desain fisik dari jaringan yang termasuk perangkat, lokasi dan instalasi kabel. Sementara topologi bersifat logis mengacu pada fakta bahwa bagaimana sebenarnya transfer data pada jaringan bertentangan dengan desainnya (Pandya, 2013).

Topologi jaringan dibangun sesuai dengan kebutuhan dan digunakan untuk menghubungkan antar komputer satu dengan komputer yang lainnya menggunakan media kabel ataupun media *wireless*. Penggunaan topologi jaringan didasarkan pada biaya, kecepatan akses data, ukuran maupun tingkat konektivitas yang akan mempengaruhi kualitas maupun efisiensi suatu jaringan.

# BAB 3 METODOLOGI

Penelitian ini diawali dengan studi literatur yang terkait dengan kajian pustaka dan dasar teori. Penelitian ini bersifat analisis dengan membandingkan kinerja dari Algoritme Floyd-Warshall dan Algoritme Johnson pada SDN yang disimulasikan menggunakan simulator *mininet*. Diawali dengan menentukan alur metode penelitian sebagai langkah yang akan ditempuh untuk menyelesaikan penelitian secara sistematis. Alur metode penelitian yang dilakukan dapat dilihat dari diagram alir pada Gambar 3.1.




**Gambar 3.1 Alur Penelitian**

## 3.1 Studi Literatur

Studi literatur merupakan tahap mencari dan mempelajari teori dasar dan referensi yang menunjang penelitian. Studi literatur pada penelitian ini dilakukan pemahaman terhadap teori-teori pendukung yang diperoleh dari berbagai sumber, seperti e-book, jurnal, paper, artikel hingga dokumentasi project. Teori-teori pendukung tersebut meliputi:

1. *Software Defined Network* (SDN) dan OpenFlow

   Studi literatur mengenai *software defined network* dan openflow dilakukan dengan mengkaji e-book, paper serta jurnal yang membahas tentang *software defined network* dan openflow.

2. Algoritme Floyd-Warshall dan Johnson

   Studi literatur mengenai konsep algoritme *routing* untuk menghitung rute terpendek menggunakan Algoritme Floyd-Warshall dan Johnson dengan mengkaji *e-book*, jurnal serta laporan penelitian terkait.

3. Pemrograman Python

   Studi literatur mengenai bahasa pemrograman *python* pada *controller* untuk menerapkan algoritme *routing* dengan mengkaji buku pedoman dan laman-laman tutorial.

## 3.2 Analisis Kebutuhan

Analisis kebutuhan ditujukan secara umum untuk melakukan analisis pada beberapa kebutuhan yang diperlukan pada penelitian. Analisis kebutuhan pada penelitian ini meliputi:

### 3.2.1 Kebutuhan Fungsional

Analisis kebutuhan fungsional yaitu menganalisa kebutuhan fungsi mengenai kemampuan dari suatu sistem. Adapun kebutuhan fungsional sistem pada penelitian ini antara lain:

- Sistem dapat menentukan jalur terpendek untuk mencapai tujuan berdasarkan sejumlah *switch* pada topologi jaringan.
- Mengukur kinerja kedua algoritme menggunakan parameter uji yang telah ditentukan.

### 3.2.2 Kebutuhan Non-Fungsional

Analisis kebutuhan non-fungsional pada penelitian ini merupakan perangkat lunak yang dibutuhkan untuk pengembangan dalam sistem. Adapun perangkat lunak yang digunakan antara lain:

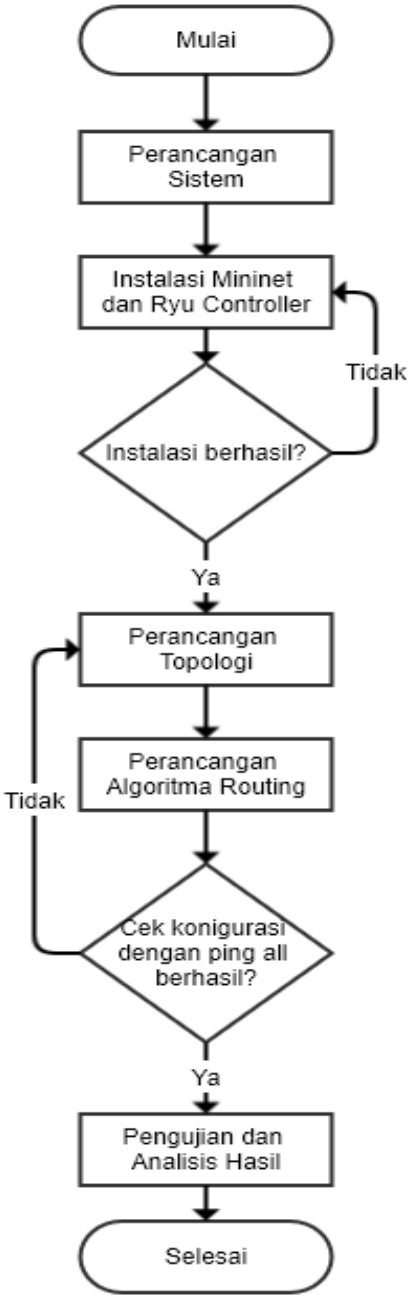
a. Perangkat keras:
   Komputer/laptop dengan spesifikasi:
   - CPU: Intel Core i3.
   - RAM: 6144 MB.
   - Hard disk: 500 GB.

b. Perangkat lunak:
   - *Ubuntu* 14.04 LTS 64 bit: sebagai sistem operasi.
   - *Ryu:* sebagai *controller* jaringan SDN.
   - *Mininet:* sebagai simulator jaringan SDN.
   - *NetworkX:* sebagai library untuk implementasikan algoritme *routing*.
   - *Ping, iperf, ps aux*: sebagai tools yang digunakan untuk pengujian.

## 3.3 Perancangan Sistem

Perancangan sistem merupakan tahapan yang dilakukan untuk membangun sistem dari penelitian setelah melakukan studi literatur dan analisis kebutuhan sistem. Tujuan dilakukan perancangan sistem agar implementasi

sistem berjalan sistematis dan terstruktur. Perancangan sistem dapat dilihat pada flowchart.



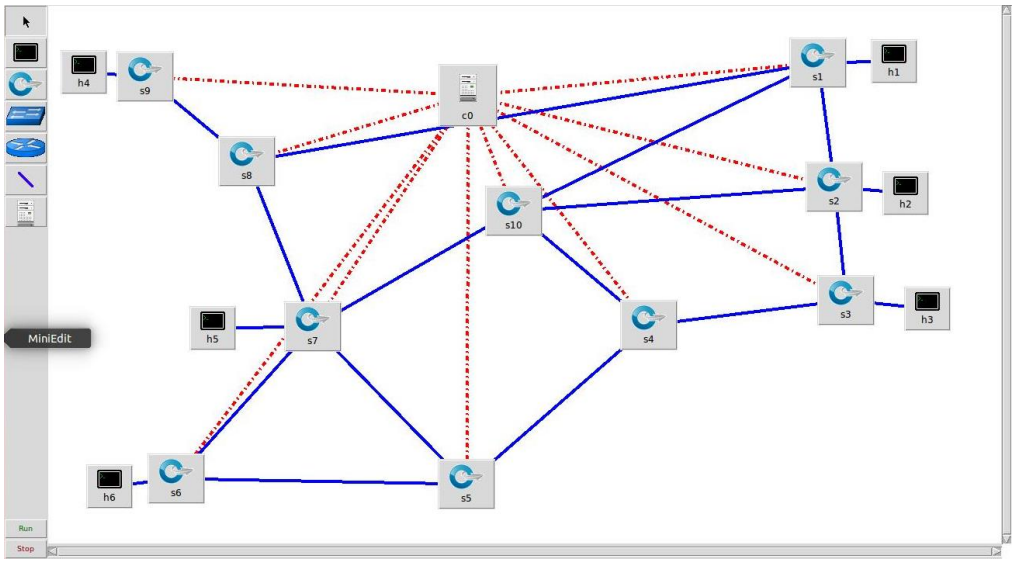

**Gambar 3.2 Flowchart umum sistem**

Berdasarkan Gambar 3.2 merupakan alur pengerjaan sistem. Dimulai dengan perancangan sistem yang meliputi perancangan topologi dan perancangan algoritme *routing*. Penelitian ini menggunakan Algoritme Floyd-Warshall dan Algoritme Johnson. Kemudian melakukan instalasi mininet sebagai simulator jaringan SDN dan ryu sebagai *controller*nya. Apabila proses instalasi telah selesai akan dilakukan proses pengecekan terlebih dahulu. Selanjutnya membuat topologi pada mininet dan melakukan perancangan algoritme *routing* dengan membuat program Floyd-Warshall dan Johnson. Program tersebut nantinya akan

diimplementasikan pada *controller* ryu. Setelah pengecekan konfigurasi dan sistem telah berjalan sesuai rancangan, berikutnya dilakukan pengujian dan analisis hasil.

### 3.3.1 Perancangan Topologi

Perancangan topologi akan menjelaskan topologi yang digunakan dalam melakukan implementasi. Perancangan topologi ini akan dibuat menggunakan *mininet* sebagai simulator. Untuk memudahkan pembuatan topologi diperlukan miniedit sebagai virtualisasinya.




**Gambar 3.3 Rancangan Topologi**

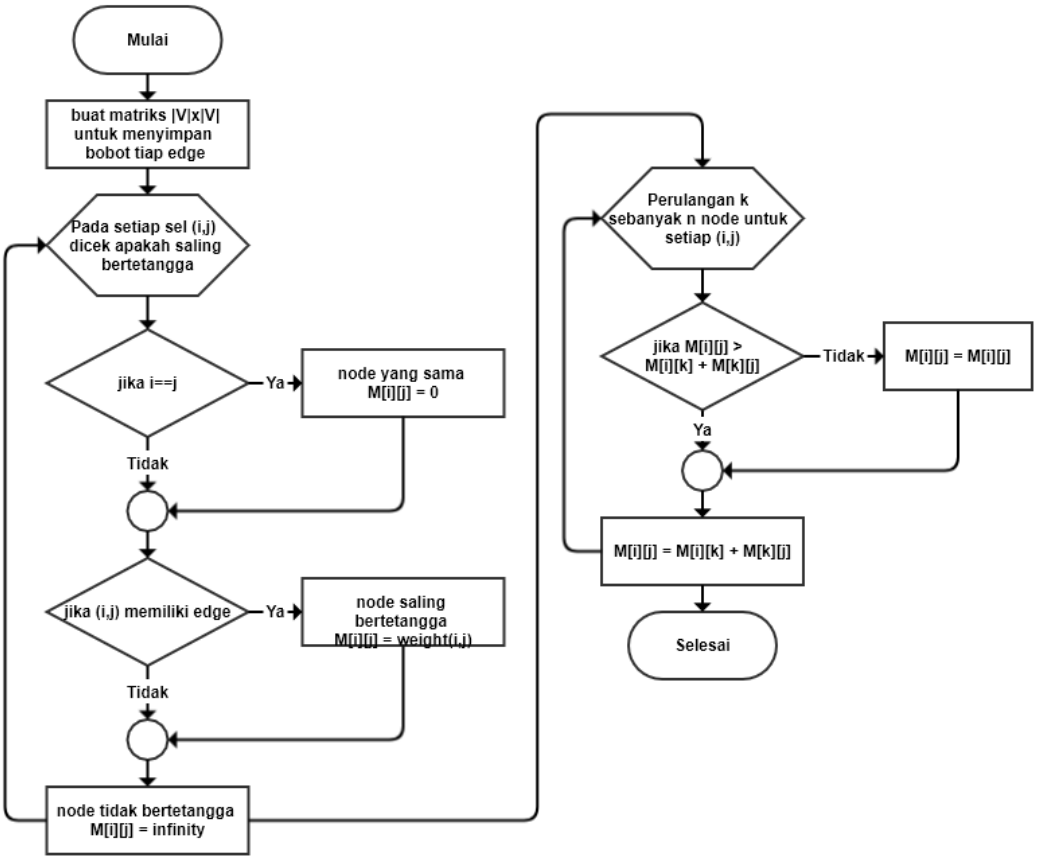
Berdasarkan Gambar 3.3 perancangan topologi pada penelitian ini akan menggunakan *switch* yang berjumlah 10 dan 6 host. Switch yang digunakan adalah *switch* openflow dan ryu sebagai *controller*nya.

### 3.3.2 Perancangan Algoritme *Routing*

Perancangan algoritme akan menjelaskan alur pencarian untuk menentukan rute terpendek pada tiap algoritme. Pada penelitian ini ada 2 algoritme yang akan diuji yaitu Algoritme Floyd-Warshall dan Algoritme Johnson. Kedua algoritme tersebut memiliki mekanisme yang berbeda dalam menentukan rute terpendek. Hasil dari perancangan ini adalah program berekstensi py yang akan dijalankan pada *controller*.

#### *3.3.2.1 Perancangan Algoritme Floyd-Warshall*

Perancangan Algoritme Floyd-Warshall akan menjelaskan mekanisme pencarian rute terpendek pada Algoritme Floyd-Warshall. Untuk menentukan rute terpendek, Algoritme Floyd-Wasrhall memulai iterasi dari titik awalnya kemudian memperpanjang *path* dengan mengevaluasi titik demi titik hingga mencapai titik tujuan dengan jumlah bobot seminimum mungkin.

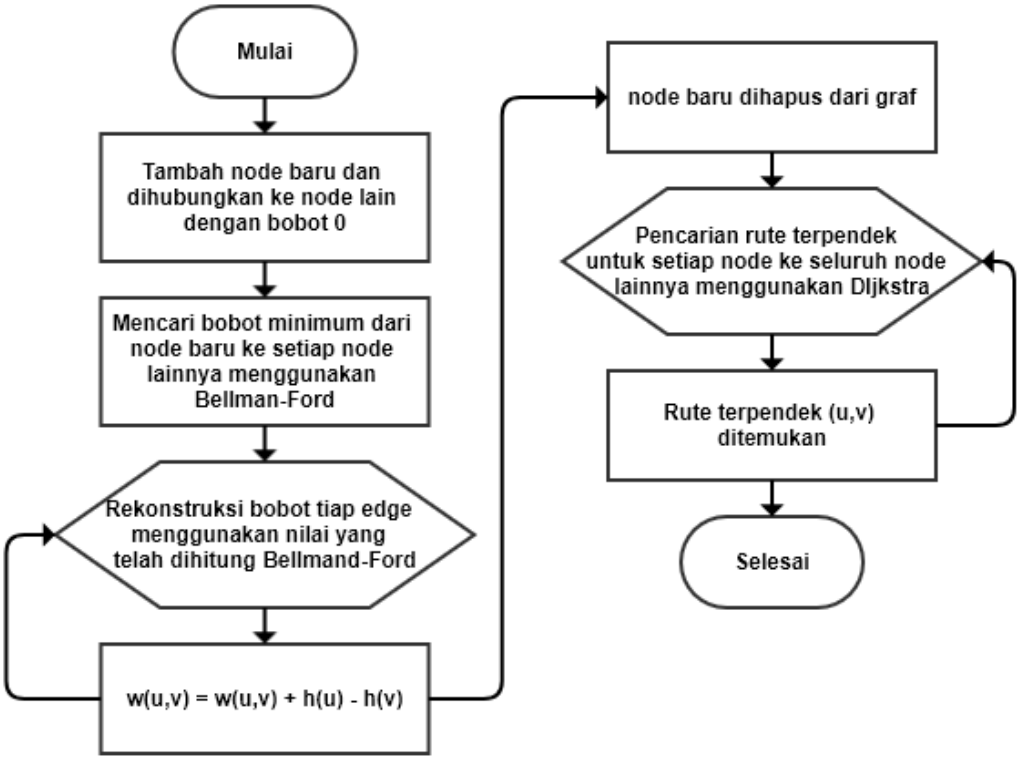



**Gambar 3.4 Flowchart Algoritme Floyd-Warshall**

Berdasarkan Gambar 3.4 merupakan mekanisme pencarian Algoritme Floyd-Warshall diawali dengan deklarasi matriks V x V yang mendeskripsikan jarak antar *node*. Untuk setiap sel (i, j) pada matriks M, apabila terdapat dua *node* saling bertetangga maka bobot pada *link* adalah weight(i, j). Sedangkan jika *node* tidak bertetangga diberikan bobot tak terhingga. Setiap *node* akan dipasangkan dengan *node* lainnya dan dihitung dengan persamaan M[i,j] > M[i,k] + M[k,j]. Apabila memenuhi syarat tersebut nilai M[i,j] akan ditukar dengan M[i,k] + M[k,j]. Proses tersebut dilakukan hingga iterasi K ke-n pada setiap M[i,j] hingga sel ke-n.

### *3.3.2.2 Perancangan Algoritme Johnson*

Perancangan Algoritme Johnson akan menjelaskan mekanisme pencarian rute terpendek pada Algoritme Johnson. Untuk menentukan rute terpendek, pada Algoritme Johnson menambahkan *node* baru pada graf. Selanjutnya adalah merekonstruksi bobot pada semua *edge node* menggunakan Algoritme Bellman-Ford sebagai subrutin. Kemudian Algoritme Dijkstra digunakan untuk mencari rute terpendek dari tiap *node* ke semua *node* lain.




**Gambar 3.5 Flowchart Algoritme Johnson**

Berdasarkan Gambar 3.5 yaitu mekanisme pencarian Algoritme Johnson diawali dengan menambahkan sebuah *node* baru yang dihubungkan pada setiap *node* lainnya dengan bobot 0. Kemudian Algoritme Bellman-Ford dijalankan untuk menghitung bobot minimum dengan *node* baru sebagai titik awal. Tahap selanjutnya adalah merekonstruksi bobot pada setiap *node* dengan persamaan w(u,v) = w(u,v) + h(u) - h(v). Lalu *node* baru tersebut dihapus dari topologi dan Algoritme Dijkstra dijalankan untuk menghitung rute terpendek dari setiap *node* ke *node* lainnya dengan bobot yang diberikan menggunakan perasamaan D(u, v) = D(u, v) + h(v) - h(u).

## 3.4 Implementasi

Pada tahap implementasi ini dilakukan dengan mengacu pada perancangan sistem yang telah dibuat. Adapun implementasi sistem sebagai berikut:

1. Melakukan instalasi simulator *mininet* dan *miniedit* pada sistem operasi *ubuntu*.
2. Melakukan instalasi *controller ryu*.
3. Melakukan instalasi *NetworkX*.
4. Membuat topologi dengan menggunakan 10 *switch* dan 6 *host*.
5. Membuat program Algoritme Floyd-Warshall dan Algoritme Johnson sebagai mekanisme *routing*.

6. Melakukan *routing* dengan masing-masing algoritme.
7. Menganalisa kinerja algoritme *routing* terhadap parameter yang diuji.
8. Membandingkan kedua algoritme dari hasil analisis yang diperoleh.

## 3.5 Pengujian

Tahap pengujian pada penelitian ini dilakukan untuk menunjukkan bahwa sistem telah mampu bekerja sesuai spesifikasi yang mendasarinya. Penelitian ini membandingkan algoritme *routing* Floyd-Warshall dan Johnson pada *software defined network*.

Pengujian akan dilakukan secara bergantian antar algoritme. Pengujian akan mengamati beberapa parameter yang telah ditentukan. Adapun parameter uji pada penelitian ini meliputi:

### 3.5.1 *Packet Loss*

*Packet Loss* didefinisikan sebagai kegagalan transmisi paket IP mencapai tujuannya. Kegagalan paket mencapai tujuan dapat diakibatkan terjadinya overload trafik dalam jaringan, kongesti, error pada media fisik atau overflow yang terjadi pada buffer (Iskandar, 2015).

### 3.5.2 *Delay*

*Delay* adalah waktu yang dibutuhkan data untuk menempuh jarak dari asal ke tujuan. *Delay* dapat dipengaruhi oleh jarak, media fisik, kongesti atau juga waktu proses yang lama (Pranata, 2016).

### 3.5.3 *Convergence Time*

*Convergence time* adalah waktu yang dibutuhkan sebuah jaringan untuk mencapai keadaan steady state pada semua *link* (Saputra, 2016). *Convergence time* yang didapat adalah waktu yang dibutuhkan algoritme *routing* untuk mencari rute baru ketika ada *link* mati pada rute sebelumnya.

### 3.5.4 CPU dan *Memory Usage*

CPU usage mengacu pada penggunaan sumber daya pemrosesan komputer atau jumlah tugas yang ditangani oleh CPU. Penggunaan CPU bervariasi tergantung pada jumlah dan jenis tugas komputasi terkelola (Techopedia, n.d.). Sedangkan *memory usage* merupakan penggunaan memori oleh rangakain tugas yang sedang dikerjakan.

## 3.6 Analisis Hasil

Dari konsep perancangan, implementasi dan pengujian terhadap sistem pada *software defined network* diperoleh hasil berupa *delay*, *packet loss*, *convergence time*, CPU dan *memory usage*. Hasil dari pengujian akan dianalisis dan dibandingkan antara kedua algoritme.

Analisis hasil yang diperoleh akan menjadi referensi dalam pemilihan algoritme *routing*. Algoritme *routing* diperlukan untuk melakukan pengiriman paket yang efisien pada *software defined network*.

## 3.7 Kesimpulan

Pada tahap ini merupakan tahap penarikan kesimpulan yang diambil dari pengujian dan analisis sistem. Selain itu pada tahap ini juga ditambahkan saran sebagai hasil akhir yang diharapkan dapat digunakan sebagai penelitian selanjutnya.

# BAB 4 PERANCANGAN & IMPLEMENTASI

Pada bab ini akan dijelaskan tentang implementasi yang dilakukan dalam pengerjaan tugas akhir. Bagian ini akan menjelaskan implementasi yang dibagi menjadi dua pembahasan yaitu instalasi dan implementasi *software defined network*.

## 4.1 Instalasi

Pada bagian ini akan dijelaskan instalasi perangkat lunak yang diperlukan dalam pengembangan sistem. Perangkat lunak yang dibutuhkan dalam system yaitu mininet sebagai simulator, ryu sebagai *controller* dan *network*X sebagai library algoritme *routing*. Langkah-langkah instalasi akan dijelaskan sebagai berikut:

### 4.1.1 Mininet

Mininet adalah sebuah simulator jaringan yang mendukung protokol openflow untuk arsitektur *software defined network*. Keunggulan mininet salah satunya mampu membangun topologi yang cukup kompleks dan sesuai keinginan perancangnya. Berikut cara instalasi mininet pada operating system ubuntu:

1. Mengunduh source code mininet pada github menggunakan perintah:

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

2. Melakukan instalasi mininet dengan menjalankan install script:

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

### 4.1.2 Ryu

Ryu merupakan framework berbasis komponen *software defined network* yang menyediakan komponen software dengan API yang membuatnya mudah untuk mengembangkan dan mengelola aplikasi. Berikut cara instalasi *controller* ryu pada operating system ubuntu:

1. Mengunduh source code ryu pada github menggunakan perintah:

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

2. Melakukan instalasi mininet dengan menggunakan perintah:

   $ python ./setup.py install

### 4.1.3 NetworkX

NetworkX merupakan paket perangkat lunak untuk menganalisa jaringan yang bersifat fleksibel dan ditulis dalam Bahasa pemrograman python. Berikut cara instalasi *controller* ryu pada operating system ubuntu:

1. Melakukan instalasi pip terlebih dahulu menggunakan perintah:

   $ sudo apt-get install python-pip

2. Melakukan instalasi *network*x menggunakan perintah pip:

   $ pip install *network*x

## 4.2 Implementasi *Software Defined Network*

Pada bagian ini akan dijelaskan *software defined network* yang diimplementasikan pada simulator mininet yang berfungsi sebagai media infrastuktur jaringan dan ryu sebagai *controller*nya.

### 4.2.1 Running Mininet dan Miniedit

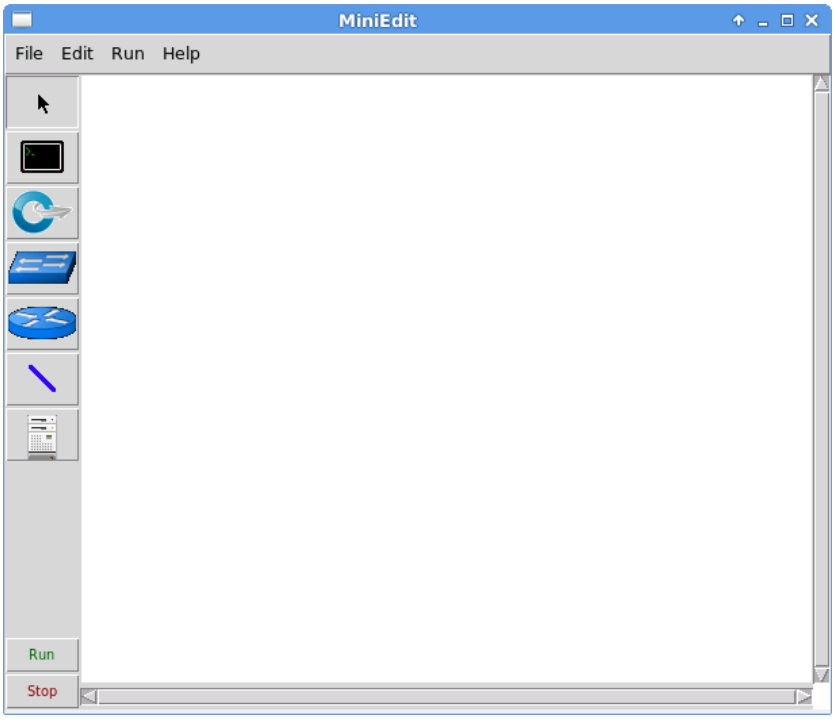
Untuk menjalankan mininet menggunakan perintah sebagai berikut:

$ sudo mn –test *ping*pair

Syntax diatas akan membuat topologi yang minimalis, menjalankan openflow *controller* reference dan melakukan perintah *ping* untuk seluruh host pada topologi.

Miniedit yang merupakan virtualisasi dari mininet dijalankan dengan perintah:

$ sudo ~/mininet/examples/miniedit.py




**Gambar 4.1 Interface Miniedit**

Gambar 4.1 merupakan interface dari miniedit. Miniedit mempermudah pengguna untuk membuat topologi. Terdapat beberapa tools pada miniedit yaitu select, host, *switch*, legacy *switch*, legacy *router*, net*link* dan *controller*. Untuk membuat topologi caranya dengan melakukan drag and drop menggunakan tools yang tersedia. Tersedia pula tombol run dan stop untuk menjalankan dan menghentikan jaringan.

### 4.2.2 Running Ryu

Untuk menjalankan *controller* ryu menggunakan perintah:

$ ryu-manager nama_file

Syntax diatas digunakan untuk melakukan upload file pada ryu *controller*. File program yang digunakan adalah file berekstensi .py. Syntax tersebut ditulis pada terminal ubuntu.

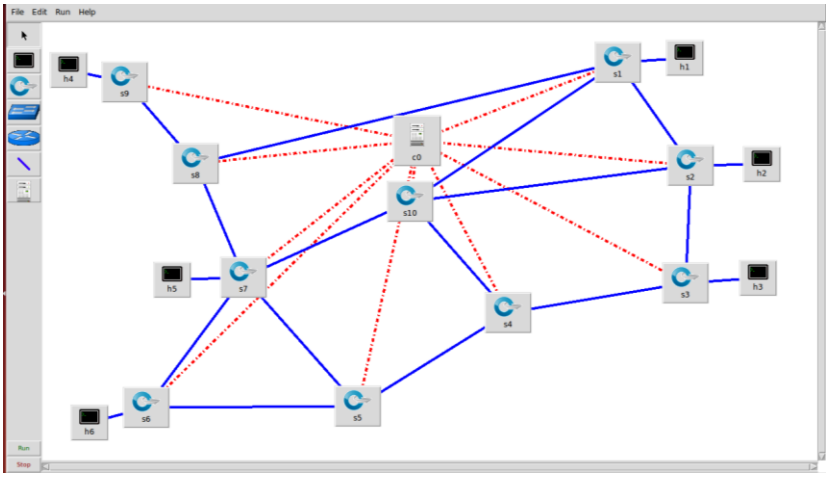
```
ilham@ilham-skripsi: ~/ryu/ryu/app/johnson
ilham@ilham-skripsi:~$ cd ~/ryu/ryu/app/johnson/
ilham@ilham-skripsi:~/ryu/ryu/app/johnson$ ryu-manager shortest_forwarding.py --
observe-links --weight=hop
lzma module is not available
Registered VCS backend: git
Registered VCS backend: hg
Registered VCS backend: svn
Registered VCS backend: bzr
loading app shortest_forwarding.py
Generating grammar tables from /usr/lib/python2.7/lib2to3/Grammar.txt
Generating grammar tables from /usr/lib/python2.7/lib2to3/PatternGrammar.txt
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app None of NetworkAwareness
creating context network_awareness
instantiating app ryu.topology.switches of Switches
instantiating app shortest_forwarding.py of ShortestForwarding
instantiating app ryu.controller.ofp_handler of OFPHandler
switch:2 connected
switch:5 connected
switch:8 connected
```

**Gambar 4.2 Running Controller Ryu**

Apabila di terminal telah tercetak tampilan seperti Gambar 4.2, maka program telah berhasil di upload pada *controller* ryu. Ketika program sudah terupload maka sistem telah siap dijalankan untuk mencari rute terpendek.

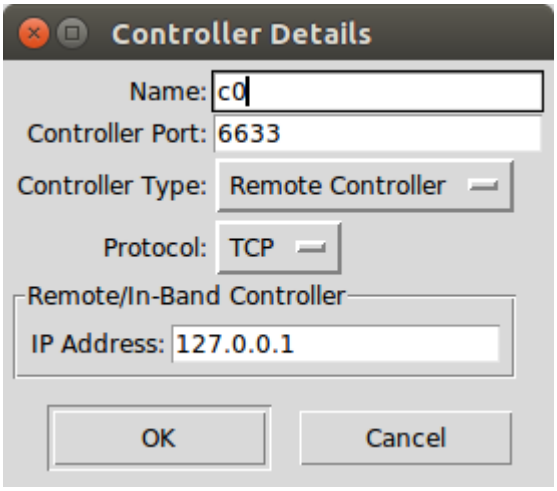
### 4.2.3 Pembuatan Topologi

Pada subbab ini akan menjelaskan pembuatan topologi yang digunakan dalam melakukan implementasi. Topologi ini akan dibuat menggunakan *mininet* sebagai simulator. Untuk memudahkan pembuatan topologi diperlukan miniedit sebagai virtualisasinya.




**Gambar 4.3 Topologi dengan 10 *switch***

Berdasarkan Gambar 4.3 perancangan topologi pada penelitian ini akan menggunakan *switch* berjumlah 10 dan 6 host. Switch yang digunakan adalah *switch* openflow dan ryu sebagai *controller*nya.




**Gambar 4.4 Setting Remote Controller**

Setelah topologi dibuat, hal yang perlu dilakukan adalah mengatur *controller* type menjadi remote *controller* seperti terlihat pada Gambar 4.4. Hal ini dimaksudkan untuk menghubungkan antara simulator mininet dengan *controller* ryu.

### 4.2.4 Implementasi *Forwarding*

Implementasi *forwarding* membahas tentang source code untuk *forwarding* paket. Informasi yang digunakan dalam proses *forwarding* akan diolah oleh *controller* dan akan disimpan pada *table* flow tiap *switch*.

Algoritme 1: Source Code *Forwarding*

```
1   def shortest_forwarding(self, msg, eth_type, ip_src, ip_dst):
2       datapath = msg.datapath
3       ofproto = datapath.ofproto
4       parser = datapath.ofproto_parser
5       in_port = msg.match['in_port']

7       result = self.get_sw(datapath.id, in_port, ip_src, ip_dst)
8       if result:
9           src_sw, dst_sw = result[0], result[1]
10          if dst_sw:
11              path = self.get_path(src_sw, dst_sw, weight=self.weight)
12              self.logger.info("[PATH]%s<-->%s: %s" % (ip_src, ip_dst,
    path))
13              if self.awareness.convergence is True:
14                  print "Convergence: ", time.time()-self.time
15              flow_info = (eth_type, ip_src, ip_dst, in_port)

16              self.install_flow(self.datapaths,
17                                self.awareness.link_to_port,
18                                self.awareness.access_table, path,
19                                flow_info, msg.buffer_id, msg.data)
20  return
```

Berdasarkan algoritme 1 maka dapat diketahui alur source code *forwarding*. Berikut penjelasan program di atas:

a. Baris 1, deklarasi prosedur shortest_*forwarding* dengan parameter berupa informasi pada paket yang masuk.

b. Baris 2-5, inisialisasi variabel berdasarkan informasi yang diperoleh dari paket yang diterima.

c. Baris 7, mengambil informasi *switch* berupa dpid, port, source ip dan destination ip.

d. Baris 8-11, jika destination ip diketahui, maka akan mengambil nilai rute yang telah ditemukan oleh algoritme *routing*.

e. Baris 12, menampilkan jalur dari source ke destination.

f. Baris 13-14, apabila nilai convergence adalah True, maka akan menampilkan *convergence time*.

g. Baris 15-19, menyimpan informasi source ip, destination ip, port, path, message data ke dalam *table* flow entry.

### 4.2.5 Implementasi Algoritme *Routing*

Pada bagian ini akan dijelaskan implementasi Algoritme Floyd-Warshall dan Johnson pada *software defined network* ke dalam sebuah program. Masing-masing algoritme memiliki mekanisme pencarian rute yang berbeda. Pada Algoritme Floyd-Warshall memiliki mekanisme pencarian yang membandingkan seluruh semua rute yang dilalui. Sedangkan Algoritme Johnson memiliki mekanisme pencarian dengan menggunakan algoritme Dijkstra dan Bellman-ford sebagai subrutin, dimana Algoritme Bellman-Ford untuk *reweighting cost* antar *node* dan Algoritme Dijkstra dieksekusi untuk mencari rute terpendek.

#### *4.2.5.1 Implementasi Algoritme Floyd-Warshall*

Implementasi Algoritme Floyd-Warshall membahas tentang Algoritme Floyd-Warshall untuk diimplementasikan ke dalam sebuah program. Program algoritme ini berdasarkan persamaan dan flowchart Algortima Floyd-Warshall.

Algoritme 2: Source Code 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)
```

Berdasarkan algoritme 2 maka dapat diketahui alur source code Algoritme Floyd-Warshall. Berikut penjelasan program di atas:

a. Baris 1, deklarasi prosedur Floyd-Warshall dengan parameter G dan weight.

b. Baris 2, import dictionary.

c. Baris 3, inisialisasi dictionary dist untuk jarak.

d. Baris 4-5, inisialisasi dist antar *node* u pada graph dengan nilai 0.

e. Baris 6, inisialisasi dictionary pred untuk predecessor.

f. Baris 7, pengecekkan graf merupakan graf berarah atau tidak

g. Baris 8-14, inisialisasi dictionary distance menjadi matrix adjacency

h. Baris 15-20, update nilai pred dan dist

i. Baris 21, return nilai dari dictionary pred dan dist.

Algoritme 3: Source Code Path Reconstruction Floyd-Warshall

```
1
2       pred, dist =
    nx.floyd_warshall_predecessor_and_distance(self.net)
3       p = list()
4         for i in pred:
5           temp = pred[i]
6           for j in temp:
7               p.append(yo[j])
8       predecessor = np.reshape(p, (len(pred),len(pred)))
9       print predecessor

10      awal = self.net.successors(src)
11      akhir = self.net.successors(dst)
12      awal = awal[0]
13      akhir = akhir[0]
14      path = list()

15      def path_reconstruct(pred, awal, akhir):
16          if pred[awal-1][akhir-1] == awal:
17              path.insert(0,awal)
18              return path
19          path_reconstruct(pred, awal, pred[awal-1][akhir-1])
20          path.append(pred[awal-1][akhir-1])
21          return path

22      def add_dst(rute,akhir):
23          rute.append(akhir)
24          rute.insert(0,src)
25          rute.append(dst)
26          return rute

27      rute = path_reconstruct(predecessor, awal, akhir)
28      path = add_dst(rute,akhir)
```

Berdasarkan algoritme 3 merupakan source code path reconstruction dari Algoritme Floyd-Warshall. Sebab Algoritme Floyd-Warshall tidak menghasilkan output berupa path, melainkan matriks berupa predecessor dan distance. Untuk

itu dibutuhkan program untuk mengubahnya menjadi path. Berikut penjelasan program di atas:

a. Baris 2, memanggil fungsi Algoritme Floyd-Warshall dengan output berupa predecessor dan distance.

b. Baris 3, deklasarasi variable p sebagai list.

c. Baris 4-7, untuk setiap nilai pada dictionary pred di insert pada list p.

d. Baris 8, membuat array multidimensi menggunakan nilai dari list p.

e. Baris 9, mencetak isi predecessor.

f. Baris 10-13, inisilaisasi variable awal dan akhir dengan suksesor dari src dan dst.

g. Baris 14, deklarasi variable path sebagai list.

h. Baris 15-21, subrutin path reconstruction. Fungsi untuk membuat rute berdasarkan predecessor hasil perhitungan Algoritme Floyd-Warshall.

i. Baris 22-26, subrutin untuk menambahkan *node* awal dan *node* tujuan pada path.

j. Baris 27, memanggil fungsi path_reconstruct dengan output berupa jalur terpendek.

k. Baris 28, memanggil fungsi add_dst dengan ouput berupa path yang telah ditambahkan *node* awal dan *node* tujuan.

### *4.2.5.2 Implementasi Algoritme Johnson*

Implementasi Algoritme Johnson akan membahas tentang Algoritme Johnson untuk diimplementasikan ke dalam sebuah program. Program algoritme ini berdasarkan persamaan dan flowchart Algortima Johnson.

Algoritme 4: Source Code Johnson

```
1  def Johnson(G, weight='weight'):
2      dist = {v: 0 for v in G}
3      pred = {v: None for v in G}
4      dist_bellman = _bellman_ford_relaxation(G, pred, dist,
           G.nodes(),weight)[1]
5      if G.is_multigraph():
6          get_weight = lambda u, v, data: (
               min(eattr.get(weight, 1) for eattr in data.values()) +
               dist_bellman[u] - dist_bellman[v])
7      else:
8          get_weight = lambda u, v, data: (data.get(weight, 1) +
               dist_bellman[u] - dist_bellman[v])
9      all_pairs = {v: _dijkstra(G, v, get_weight, paths={v: [v]})[1]
   for v in G}
10     return all_pairs
```

Berdasarkan algoritme 4 maka dapat diketahui alur source code Algoritme Johnson. Berikut penjelasan program diatas:

a. Baris 1, deklarasi prosedur Johnson dengan parameter G dan weight.

b. Baris 2, inisialisasi dictionary dist dengan 0 untuk setiap v pada graf.

c. Baris 3, inisialisasi dictionary pred dengan none untuk setiap v pada graf.

d. Baris 4, menghitung jarak terpendek menggunakan sub algoritme bellman-ford.

e. Baris 5-8, pengecekan graf merupakan multigraf atau tidak dan mengambil bobot baru yang telah dihitung oleh algoritme bellman-ford.

f. Baris 9, menghitung jarak terpendek menggunakan sub algoritme dijkstra pada graf yang telah diubah bobotnya.

g. Baris 10, return distance dan rute terpendek.

# BAB 5 PENGUJIAN DAN ANALISIS

Bab pengujian dan analisis membahas tentang proses pengujian pada penelitian yang dilakukan yaitu tentang "Analisa Perbandingan Kinerja Algoritme Floyd-Warshall dan Algoritme Johnson Untuk Penentuan Rute Terpendek Pada *Software Defined Network*". Ada dua pembahasan pada bab ini yang pertama adalah pengujian. Pengujian berisi pengujian Algoritme Floyd-Warshall dan Algoritme Johnson berdasarkan parameter uji. Pembahasan yang kedua adalah analisis hasil pengujian dari masing-masing algoritme dan perbandingan antar keduanya.

## 5.1 Pengujian

Pada bagian ini akan dilakukan pengujian penentuan rute terpendek dan beberapa parameter uji yaitu *packet loss*, *delay*, *convergence time*, CPU dan memori usage. Hasil dari setiap pengujian diambil dari pengujian ketika program pertama kali dijalankan, karena ketika pertama kali dijalankan program akan melakukan mekanisme pencarian *routing table* dan *forwarding*.

### 5.1.1 Pengujian Pemilihan Rute

**1. Tujuan**

Pengujian pemilihan rute ini bersifat fungsionalitas karena pada pengujian ini hanya akan mengamati dan membuktikan bahwa sistem yang menerapkan Algoritme Floyd-Warshall dan Algoritme Johnson telah dapat menemukan rute terpendek dari sejumlah *node* pada topologi.

**2. Skema**

Pada pengujian rute terpendek telah ditentukan bobot tiap-tiap *link* yang nantinya akan dihitung oleh algoritme routing untuk mencari rute terpendek. Pada pengujian pemilihan rute terdapat 3 skenario pengujian, antara lain:

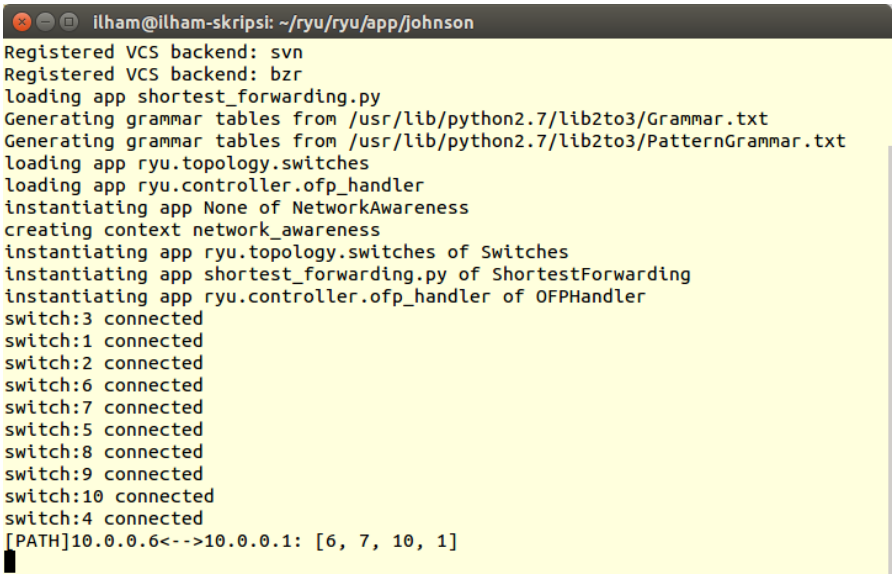
a. Berdasarkan total bobot dan jumlah *hop* yang dilewati sama.

b. Berdasarkan total bobot sama namun jumlah *hop* yang dilewati berbeda.

c. Berdasarkan total bobot dan jumlah *hop* yang dilewati berbeda.

**3. Prosedur**

Berdasarkan Gambar 5.1 merupakan proses pengujian pemilihan rute. Pengujian ini menggunakan *tool ping* untuk melakukan proses komunikasi antara *node* asal dengan *node* tujuan. Kemudian pada terminal akan ditampilkan rute yang dilewati oleh paket dari *node* asal ke *node* tujuan.




**Gambar 5.1 Pengujian Pemilihan Rute**

**Tabel 5.1 Hasil Pengujian Pemilihan Rute Floyd-Warshall**

| No | Switch | | Pilihan rute | Rute yang dilewati | Bobot antar *switch* | Total bobot |
|---|---|---|---|---|---|---|
| | Asal | Tujuan | | | | |
| 1 | 6 | 1 | 6->7->8->1 | 6->7->8->1 | 6->7=2, 7->8=3, 7->10=2, 8->1=2, 10->1=2 | 6 |
| | | | 6->7->10->1 | | | |
| 2 | 4 | 3 | 9->8->1->2->3 | 9->8->1->2->3 | 1->2=3, 2->3=2, 4->3=2, 7->10=1, 8->1=3, 8->7=2, 9->8=2, 10->4=3 | 10 |
| | | | 9->8->7->10->4->3 | | | |
| 3 | 5 | 2 | 7->10->2 | 7->5->4->3->2 | 3->2=2, 4->3=2, 5->4=2, 7->5=2, 7->10=5, 10->2=4 | 9 |
| | | | 7->5->4->3->2 | | | 8 |

Berdasarkan Tabel 5.1 merupakan hasil pengujian pemilihan rute Floyd-Warshall. Pada skenario pertama dimana total bobot dan jumlah *hop* yang dilewati sama, Floyd-Warshall memilih rute 6->7->8->1 dengan total bobot 6. Pada skenario kedua dimana total bobot sama namun jumlah *hop* yang dilewati berbeda, Floyd-Warshall memilih rute yang jumlah *hop*nya lebih sedikit yaitu 9->8->1->2->3 dengan total bobot 10. Kemudian pada skenario ketiga dimana total bobot dan jumlah *hop* yang dilewati berbeda, Floyd-Warshall memilih rute yang total bobotnya lebih sedikit yaitu rute 7->5->4->3->2 meski rute lainnya memiliki jumlah *hop* yang lebih sedikit.

**Tabel 5.2 Hasil Pengujian Pemilihan Rute Johnson**

| No | Switch | | Pilihan rute | Rute yang dilewati | Bobot antar *switch* | Total bobot |
|---|---|---|---|---|---|---|
| | Asal | Tujuan | | | | |
| 1 | 6 | 1 | 6->7->8->1 | 6->7->10->1 | 6->7=2, 7->8=3, 7->10=2, 8->1=2, 10->1=2 | 6 |
| | | | 6->7->10->1 | | | |
| 2 | 4 | 3 | 9->8->1->2->3 | 9->8->1->2->3 | 1->2=3, 2->3=2, 4->3=2, 7->10=1, 8->1=3, 8->7=2, 9->8=2, 10->4=3 | 10 |
| | | | 9->8->7->10->4->3 | | | |
| 3 | 5 | 2 | 7->10->2 | 7->5->4->3->2 | 3->2=2, 4->3=2, 5->4=2, 7->5=2, 7->10=5, 10->2=4 | 9 |
| | | | 7->5->4->3->2 | | | 8 |

Berdasarkan Tabel 5.2 merupakan hasil pengujian pemilihan rute Johnson. Pada skenario pertama dimana total bobot dan jumlah *hop* yang dilewati sama, Johnson memilih rute 6->7->10->1 dengan total bobot 6. Pada skenario kedua dimana total bobot sama namun jumlah *hop* yang dilewati berbeda, Johnson memilih rute yang jumlah *hop*nya lebih sedikit yaitu 9->8->1->2->3 dengan total bobot 10. Kemudian pada skenario ketiga dimana total bobot dan jumlah *hop* yang dilewati berbeda, Johnson memilih rute yang total bobotnya lebih sedikit yaitu rute 7->5->4->3->2 meski rute lainnya memiliki jumlah *hop* yang lebih sedikit.

### 5.1.2 Pengujian *Packet Loss*

**1. Tujuan**

Tujuan dari pengujian *packet loss* adalah untuk mengetahui seberapa banyak packet yang hilang selama proses pengiriman data. Semakin sedikit *packet loss* yang terjadi, maka menunjukkan semakin baik pula sistem.

**2. Skema**

Pada pengujian *packet loss*, nilai bandwidth ditentukan dan menjadi acuan untuk uji coba *packet loss*. Bandwidth disetting mulai dari ukuran 125, 250, 375, 500, 625, 750, 875 dan 1000 Mpbs. Tool *iperf* digunakan untuk melakukan pengujian *packet loss*.

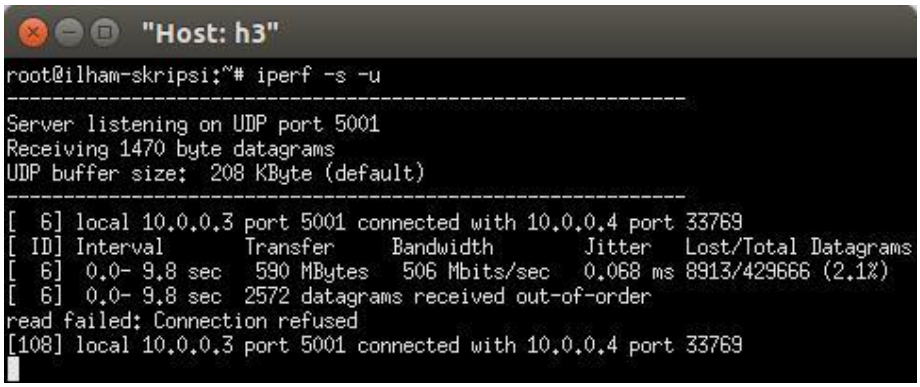
Pengujian dilakukan dengan tiga skenario dengan *node* asal dan *node* tujuan yang berbeda. Host h4, h5 dan h6 menjadi *node* asal (client) dengan masing-masing *node* tujuannya (server) adalah host h3, h2 dan h1. Seluruh hasil pengujian *packet loss* terlampir pada lampiran 1, 2 dan 3.

**3. Prosedur**

**Tabel 5.3 Syntax *running iperf***

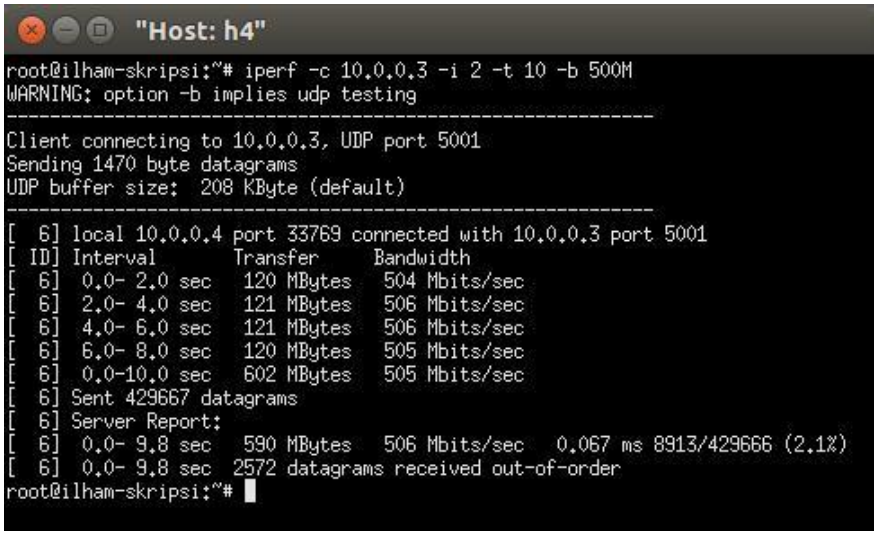
| Host | Syntax |
|---|---|
| Server | Iperf -s -u |
| Client | Iperf -c IP_server -i interval -t waktu_kirim -b ukuran_bandwidth |

Sebelum pengujian dilakukan maka terlebih dahulu menjalankan terminal pada masing-masing host yang menjadi titik awal dan titik tujuan. Berdasarkan Tabel 5.3 merupakan syntax untuk melakukan uji parameter *packet loss*. Pada pengujian ini telah ditentukan satu host yang menjadi server sebagai host tujuan. Host yang bertindak sebagai server dijalankan terlebih dahulu. Nilai *packet loss* yang diambil adalah nilai rata-rata yang tercetak pada terminal. Pada pengujian ini menggunakan interval waktu 2 detik dan waktu kirim selama 10 detik.




**Gambar 5.2 Pengujian *packet loss* sisi server**

Berdasarkan Gambar 5.2 merupakan proses pengujian *packet loss* dari sisi server. Protokol yang digunakan adalah protokol UDP. Setelah proses *iperf* selesai dapat dilihat report berbentuk tabel, dimana kolom datagram adalah *packet loss* yang terjadi.




**Gambar 5.3 Pengujian *packet loss* sisi client**

Pada Gambar 5.3 adalah sisi client dari proses pengujian *packet loss*. Host h4 yang bertindak sebagai client melakukan *iperf* pada host h3 yang merupakan server. Pengujian *packet loss* menggunakan *iperf* dilakukan selama 10 detik.

**Tabel 5.4 Hasil Pengujian *Packet Loss* Floyd-Warshall**

| No | Parameter Uji | |
|---|---|---|
| | Bandwidth (Mbps) | *Packet Loss* (%) |
| 1 | 125 | 5,03 |
| 2 | 250 | 4,1 |
| 3 | 375 | 2,3 |
| 4 | 500 | 2,5 |
| 5 | 625 | 2,47 |
| 6 | 750 | 1,47 |
| 7 | 875 | 0,68 |
| 8 | 1000 | 0,06 |
| **Rata-rata** | | **2,33** |

Pada pengujian *packet loss* Floyd-Warshall dapat diketahui bahwa semakin besar bandwidth yang diberikan maka semakin kecil pula kemungkinan terjadinya paket loss. Berdasarkan Tabel 5.4 merupakan nilai rata-rata *packet loss* dari 3 skenario berbeda. *Packet loss* yang terjadi berkisar diantara 5,03% – 0,06% dalam rentang bandwidth 125 – 1000 Mbps. Dengan rata-rata keseluruhan *packet loss* sebesar 2,33%.

**Tabel 5.5 Hasil Pengujian *Packet Loss* Johnson**

| No | Parameter Uji | |
|---|---|---|
| | Bandwidth (Mbit) | *Packet Loss* (%) |
| 1 | 125 | 5,67 |
| 2 | 250 | 3,63 |
| 3 | 375 | 3,57 |
| 4 | 500 | 2,2 |
| 5 | 625 | 2,3 |
| 6 | 750 | 1,7 |
| 7 | 875 | 0,69 |
| 8 | 1000 | 0,06 |
| **Rata-rata** | | **2,48** |

Pada pengujian *packet loss* Johnson dapat diketahui bahwa semakin besar bandwidth yang diberikan maka semakin kecil pula kemungkinan terjadinya paket loss. Berdasarkan Tabel 5.5 merupakan nilai rata-rata *packet loss* dari 3 skenario berbeda. *Packet loss* yang terjadi berkisar diantara 5,67% – 0,06% dalam rentang bandwidth 125 – 1000 Mbps. Dengan rata-rata keseluruhan *packet loss* sebesar 2,48%.

### 5.1.3 Pengujian *Delay*

**1. Tujuan**

Tujuan dari pengujian *delay* adalah untuk mengetahui waktu yang dibutuhkan paket untuk menempuh jarak dari *node* asal ke *node* tujuan. Jika nilai *delay* yang diperoleh semakin kecil, maka semakin baik pula sistem.

**2. Skema**

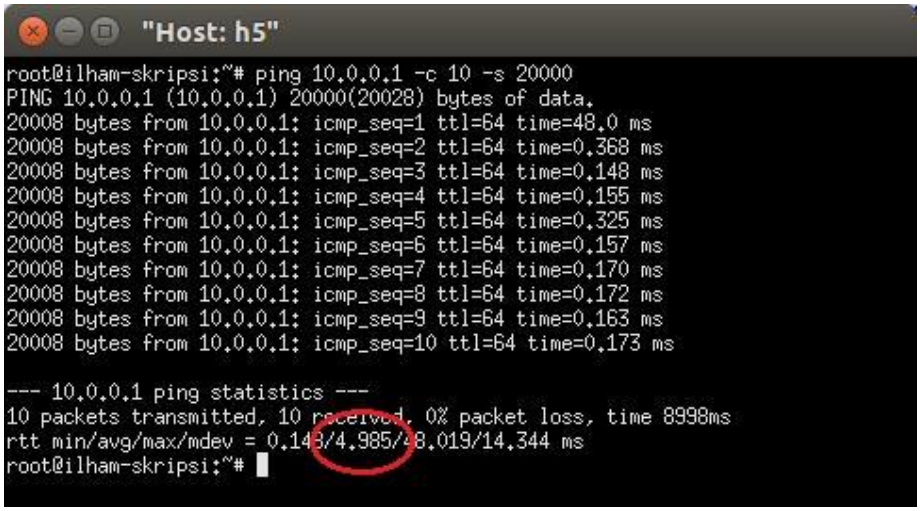
Pada pengujian *delay* menggunakan *ping* sebagai tool untuk melakukan uji coba. Ukuran paket yang dikirim pada saat melakukan command *ping* yaitu 100, 500, 1000, 2500, 5000, 10000, 15000, 20000, 25000, 30000 byte.

Pengujian dilakukan dengan tiga skenario dengan *node* asal dan *node* tujuan yang berbeda. Host h4, h5 dan h6 menjadi *node* asal dengan masing-masing *node* tujuannya adalah host h3, h2 dan h1. Seluruh hasil pengujian *delay* terlampir pada lampiran 4, 5 dan 6.

**3. Prosedur**

$ *ping* alamat_IP_tujuan -c banyaknya_paket_dikirim -s ukuran_paket

Syntax di atas digunakan untuk melakukan uji parameter *delay*. Host yang menjadi titik awal melakukan *ping* ke host tujuan. Banyaknya pengiriman pada pengujian ini adalah 10. Nilai *delay* yang diambil adalah nilai rata-rata yang tercetak pada terminal.




**Gambar 5.4 Pengujian *delay***

Berdasarkan Gambar 5.4 merupakan proses pengujian *delay* menggunakan tool *ping*. Host h5 yang melakukan *ping* ke alamat IP 10.0.0.1 sebanyak sepuluh kali dengan besar paket 20008 bytes. Hasil yang diambil adalah rata-rata *delay* yang ditunjukkan oleh lingkaran merah.

**Tabel 5.6 Hasil Pengujian *Delay* Floyd Warshall**

| No | Parameter Uji | |
|---|---|---|
| | Ukuran Paket (byte) | *Delay* (ms) |
| 1 | 100 | 1,22 |
| 2 | 500 | 1,35 |
| 3 | 1000 | 1,28 |
| 4 | 2500 | 1,68 |
| 5 | 5000 | 2,69 |
| 6 | 10000 | 3,68 |
| 7 | 15000 | 5,25 |
| 8 | 20000 | 6,49 |
| 9 | 25000 | 7,87 |
| 10 | 30000 | 9,33 |
| **Rata-rata** | | **4,08** |

Dari Tabel 5.6 merupakan nilai rata-rata *delay* Floyd-Warshall dari 3 skenario berbeda. Berdasarkan hasil pengujian yang diperoleh bahwa perubahan ukuran paket yang dikirim tidak terlalu mempengaruhi *delay*. Nilai *delay* terjadi kenaikkan, akan tetapi tidak terlalu signifikan. Pada seluruh pengujian saat paket diberi beban 100 – 30000 byte, *delay* yang diperoleh dibawah 10 ms.

**Tabel 5.7 Hasil Pengujian *Delay* Johnson**

| No | Parameter Uji | |
|---|---|---|
| | Ukuran Paket (byte) | *Delay* (ms) |
| 1 | 100 | 1,17 |
| 2 | 500 | 1,05 |
| 3 | 1000 | 1,23 |
| 4 | 2500 | 1,73 |
| 5 | 5000 | 2,81 |
| 6 | 10000 | 3,4 |
| 7 | 15000 | 5,88 |

| | | |
|---|---|---|
| 8 | 20000 | 6,52 |
| 9 | 25000 | 7,46 |
| 10 | 30000 | 8,9 |
| **Rata-rata** | | **4,02** |

Dari Tabel 5.7 merupakan nilai rata-rata *delay* Johnson dari 3 skenario berbeda. Berdasarkan hasil pengujian yang diperoleh bahwa perubahan ukuran paket yang dikirim tidak terlalu mempengaruhi *delay*. Nilai *delay* terjadi kenaikkan, akan tetapi tidak terlalu signifikan. Pada seluruh pengujian saat paket diberi beban 100 – 30000 byte, *delay* yang diperoleh dibawah 10 ms.

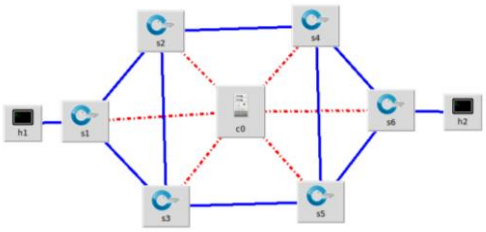
### 5.1.4 Pengujian *Convergence Time*

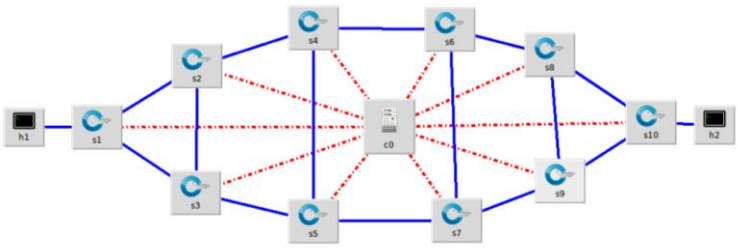
**1. Tujuan**

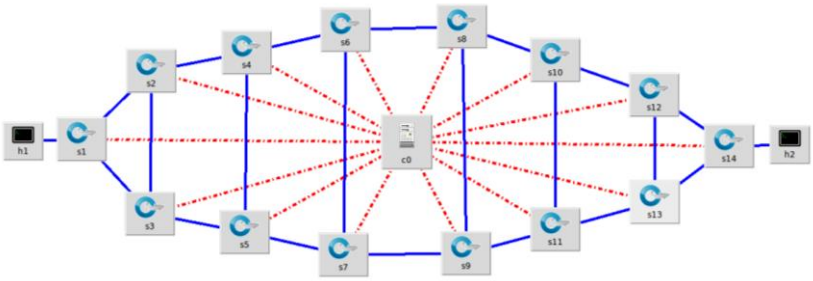
Pengujian *convergence time* dilakukan untuk mengetahui waktu yang dibutuhkan sebuah jaringan untuk mencapai keadaan steady state. Apabila hasil *convergence time* yang diperoleh semakin kecil, maka menunjukkan bahwa algoritme *routing* yang diterapkan pada sistem dapat menentukan rute dengan cepat.

**2. Skema**

Untuk pengujian *convergence time* dimulai dengan melakukan *ping* dari salah satu host ke host lainnya. Pada saat proses *ping* berjalan, dilakukan pemutusan salah satu *link* pada jalur dari host asal ke host tujuan. *Convergence time* yang didapat adalah waktu yang dibutuhkan algoritme *routing* untuk mencari rute baru ketika ada *link* mati pada rute sebelumnya. Tool untuk melakukan pengujian *convergence time* adalah *ping*.





**Gambar 5.5 Topologi 6, 10 dan 14 *switch***

Pengujian ini dilakukan pada tiga topologi dengan jumlah *switch* yang berbeda. Masing-masing topologi berjumlah 6, 10 dan 14 *switch* seperti yang dapat dilihat pada Gambar 5.5. Tiap topologi dilakukan pengujian sebanyak 10 kali.

**3. Prosedur**

Berdasarkan Gambar 5.6 merupakan proses pengujian *convergence time*. Pengujian ini dilakukan menggunakan tool *ping*. Host h6 melakukan *ping* ke host h1. Kemudian ditengah proses *ping* dilakukan pemutusan *link* seperti yang ditunjukkan oleh angka 1. Kemudian program akan melakukan proses *routing* kembali dan diperoleh rute baru dan *convergence time* yang ditunjukkan oleh angka 2.

```
ilham@ilham-skripsi: ~/ryu/ryu/app/floyd
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app None of NetworkAwareness
creating context network_awareness
instantiating app ryu.topology.switches of Switches
instantiating app shortest_forwarding.py of ShortestForwarding
instantiating app ryu.controller.ofp_handler of OFPHandler
switch:5 connected
switch:2 connected
switch:8 connected
switch:3 connected
switch:9 connected
switch:1 connected
switch:4 connected
switch:7 connected
switch:10 connected
switch:6 connected
[PATH]10.0.0.6<-->10.0.0.1: [6, 7, 8, 1]
link delete
Link: Port<dpid=8, port_no=3, DOWN> to Port<dpid=1, port_no=2, LIVE>     1
link delete
Link: Port<dpid=1, port_no=2, LIVE> to Port<dpid=8, port_no=3, LIVE>
[PATH]10.0.0.6<-->10.0.0.1: [6, 7, 10, 1]
Convergence:  42.6105458736
```

**Gambar 5.6 Pengujian *Convergence Time***

**Tabel 5.8 Hasil Pengujian *Convergence Time* Floyd-Warshall**

| No | Jumlah Switch | Waku Konvergensi (s) |
|---|---|---|
| 1 | 6 | 34,54 |
| 2 | 10 | 36,13 |
| 3 | 14 | 38,53 |

Berdasarkan Tabel 5.8 hasil pengujian *convergence time* yang diperoleh dapat diketahui bahwa semakin banyak *switch* pada topologi maka waktu yang dibutuhkan untuk menemukan rute akan semakin lama. Rata-rata *convergence time* yang dibutuhkan Algoritme Floyd-Warshall dari seluruh skenario untuk menemukan rute setelah dilakukan pemutusan *link* adalah 36,4 detik.

**Tabel 5.9 Hasil Pengujian *Convergence Time* Johnson**

| No | Jumlah Switch | Waku Konvergensi (s) |
|---|---|---|
| 1 | 6 | 28,91 |
| 2 | 10 | 35,89 |
| 3 | 14 | 39,34 |

Berdasarkan Tabel 5.9 hasil pengujian *convergence time* yang diperoleh dapat diketahui bahwa semakin banyak *switch* pada topologi maka waktu yang dibutuhkan untuk menemukan rute akan semakin lama. Rata-rata *convergence time* yang dibutuhkan Algoritme Johnson dari seluruh skenario untuk menemukan rute setelah dilakukan pemutusan *link* adalah 34,71 detik.

### 5.1.5 Pengujian CPU & *Memory Usage*

**1. Tujuan**

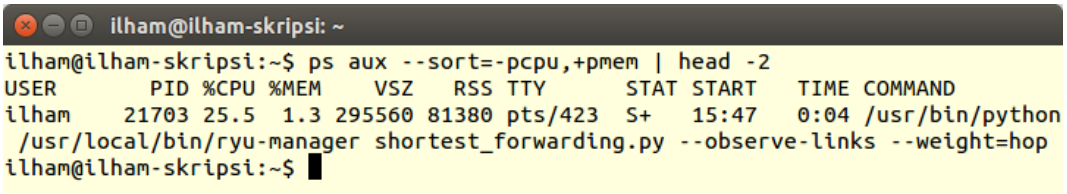
Pengujian CPU & *memory usage* dilakukan untuk mengamati seberapa besar program algoritme *routing* mengonsumsi sumber daya yang tersedia. Apabila hasil CPU & *memory usage* yang diperoleh semakin kecil, maka menunjukkan bahwa algoritme *routing* yang diterapkan pada sistem pun semakin baik.

**2. Skema**

Pengujian ini dilakukan dengan mengamati CPU & *memory usage* ketika program pertama kali dijalankan. Sebab pada saat pertama kali dijalankan,

program akan mencari rute berdasarkan informasi topologi jaringan yang diperoleh ke dalam bentuk graph menggunakan library *networkX*.

**3. Prosedur**




**Gambar 5.7 Pengujian CPU & *Memory Usage***

Berdasarkan Gambar 5.7 adalah proses pengujian CPU & *memory usage*. Tool ps aux digunakan sebagai tool untuk mengamati besarnya CPU & *memory* yang dikonsumsi oleh program. Kolom %CPU merupakan nilai dari CPU usage, sedangkan %MEM adalah nilai dari *memory usage*.

**Tabel 5.10 Hasil Pengujian CPU & *Memory Usage* Floyd-Warshall**

| No | CPU Usage (%) | *Memory Usage* (%) |
|---|---|---|
| 1 | 22,48 | 1,3 |
| 2 | 21,76 | 1,3 |
| 3 | 21,44 | 1,3 |
| 4 | 21,46 | 1,3 |
| 5 | 21,8 | 1,3 |
| 6 | 22,50 | 1,3 |
| 7 | 21,86 | 1,3 |
| 8 | 21,93 | 1,3 |
| 9 | 21,49 | 1,3 |
| 10 | 21,12 | 1,3 |
| **Rata-rata** | **21,78** | **1,3** |

Berdasarkan Tabel 5.10 hasil dari sepuluh kali percobaan dapat diketahui bahwa konsumsi CPU menggunakan Algoritme Floyd-Warshall berkisar diantara 21,12% – 22,5%. Dengan nilai rata-rata CPU usage 21,78%. Sedangkan untuk *memory usage* tidak terjadi perubahan, nilai *memory usage* konsisten berada pada angka 1,3% untuk seluruh percobaan.

**Tabel 5.11 Hasil Pengujian CPU & *Memory Usage* Johnson**

| No | CPU Usage (%) | *Memory Usage* (%) |
|---|---|---|
| 1 | 23,06 | 1,3 |
| 2 | 23,12 | 1,3 |

| | | |
|---|---|---|
| 3 | 23,4 | 1,3 |
| 4 | 23,42 | 1,3 |
| 5 | 23,5 | 1,3 |
| 6 | 23,98 | 1,3 |
| 7 | 23,57 | 1,3 |
| 8 | 24,07 | 1,3 |
| 9 | 23,64 | 1,3 |
| 10 | 23,94 | 1,3 |
| **Rata-rata** | **23,84** | **1,3** |

Berdasarkan Tabel 5.11 hasil dari lima kali percobaan diperoleh dapat diketahui bahwa konsumsi CPU menggunakan Algoritme Johnson berkisar diantara 23,06% – 24,07%. Dengan nilai rata-rata CPU usage 23,84%. Sedangkan untuk *memory usage* tidak terjadi perubahan, nilai *memory usage* konsisten berada pada angka 1,3% untuk seluruh percobaan.

## 5.2 Analisis Hasil Pengujian

Pada subbab ini akan membahas analisis hasil dari pengujian masing-masing algoritme. Kemudian hasil analisis dari masing-masing algoritme akan dibandingkan untuk memperoleh algoritme mana yang lebih baik berdasarkan parameter yang telah diuji.

### 5.2.1 Analisis Hasil Pengujian Pemilihan Rute

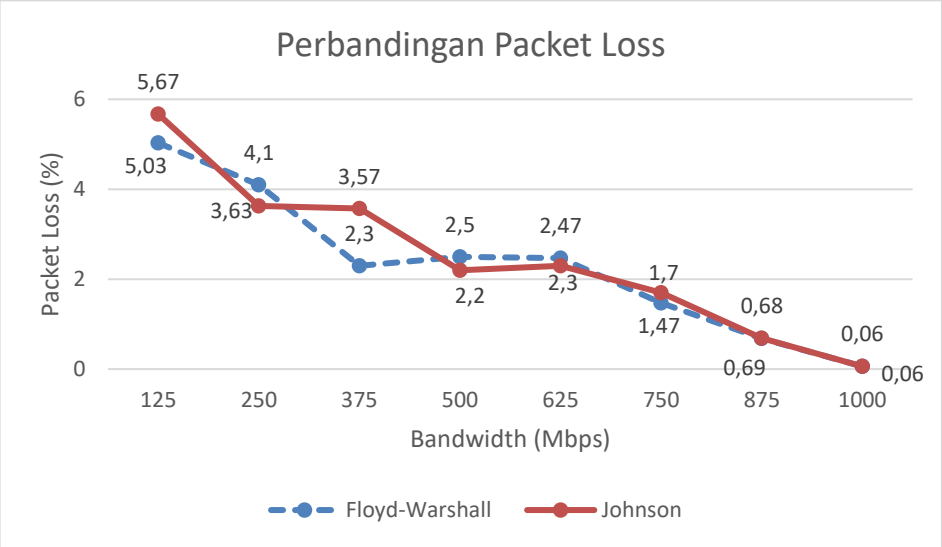
**Tabel 5.12 Perbandingan Pemilihan Rute**

| No | Switch | | Pilihan rute | Total bobot | Floyd-Warshall | Johnson |
|---|---|---|---|---|---|---|
| | Asal | Tujuan | | | | |
| 1 | 6 | 1 | 6->7->8->1 | 6 | 6->7->8->1 | 6->7->10->1 |
| | | | 6->7->10->1 | | | |
| 2 | 4 | 3 | 9->8->1->2->3 | 10 | 9->8->1->2->3 | 9->8->1->2->3 |
| | | | 9->8->7->10->4->3 | | | |
| 3 | 5 | 2 | 7->10->2 | 9 | 7->5->4->3->2 | 7->5->4->3->2 |
| | | | 7->5->4->3->2 | 8 | | |

Berdasarkan Tabel 5.12 merupakan hasil pengujian pemilihan rute Algoritme Floyd-Warshall dan Algoritme Johnson. Pada skenario pertama dimana total bobot dan jumlah *hop* yang dilewati sama, masing-masing algoritme memilih rute yang berbeda. Floyd-Warshall memilih rute 6->7->8->1, sedangkan Johnson dengan rute 6->7->10->1. Pada skenario kedua dimana total bobot sama namun jumlah *hop* yang dilewati berbeda, baik Floyd-Warshall maupun Johnson memilih rute 9->8->1->2->3 dengan total bobot 10. Hal itu dikarenakan kedua rute memiliki total bobot yang sama, sehingga keputusan pemilihan rute akan dilakukan dengan melihat jumlah *hop* yang lebih sedikit. Kemudian pada skenario ketiga dimana total bobot dan jumlah *hop* yang dilewati berbeda, Floyd-Warshall dan Johnson memilih rute 7->5->4->3->2. Meski rute 7->10->2 memiliki jumlah *hop* yang lebih sedikit, akan tetapi total bobotnya lebih besar yaitu 9 sehingga tidak dipilih oleh algoritme. Karena pemilihan rute terpendek dilakukan dengan membandingkan total nilai bobot yang paling minimum.

### 5.2.2 Analisis Hasil Pengujian *Packet Loss*

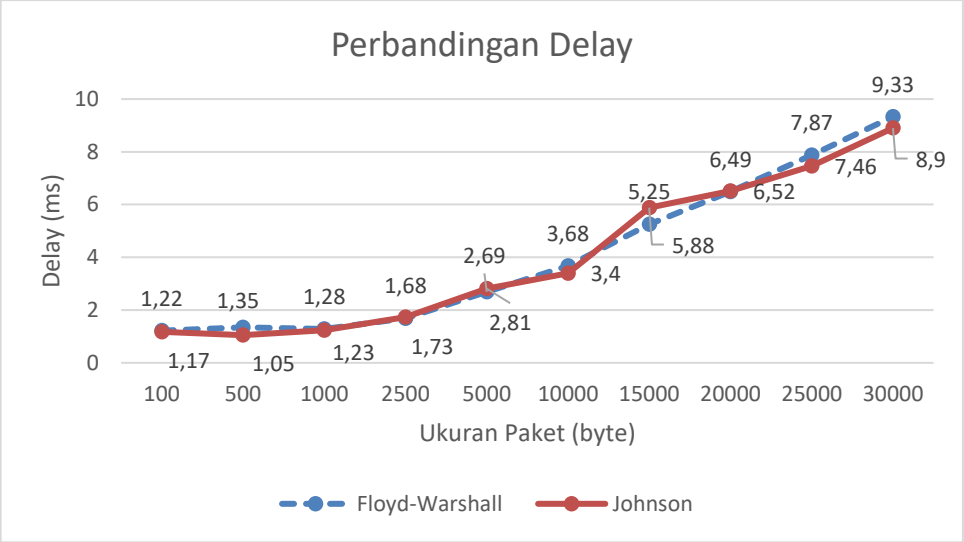
Berdasarkan Gambar 5.8 merupakan grafik hasil pengujian *packet loss* Algoritme Floyd-Warshall dan Algoritme Johnson. Algoritme Floyd-Warshall sedikit lebih unggul karena *packet loss* yang terjadi di atas 3% hanya saat diberi bandwidth 125 dan 250 Mbps. Sedangkan Algoritme Johnson saat bandwidth 375 Mbps, *packet loss* yang terjadi masih di atas 3%. Namun ketika bandwidth dinaikkan menjadi 500 hingga 1000 Mbps, masing-masing algoritme sama-sama mengalami penurunan *packet loss* di bawah 3%. Kedua algoritme mendapatkan hasil dengan perbedaan yang tidak terlalu signifikan, karena setelah rute ditemukan minim terjadi *collision* atau *congestion* yang merupakan faktor yang dapat mempengaruhi *packet loss*.




**Gambar 5.8 Grafik Perbandingan *Packet Loss***

### 5.2.3 Analisis Hasil Pengujian *Delay*

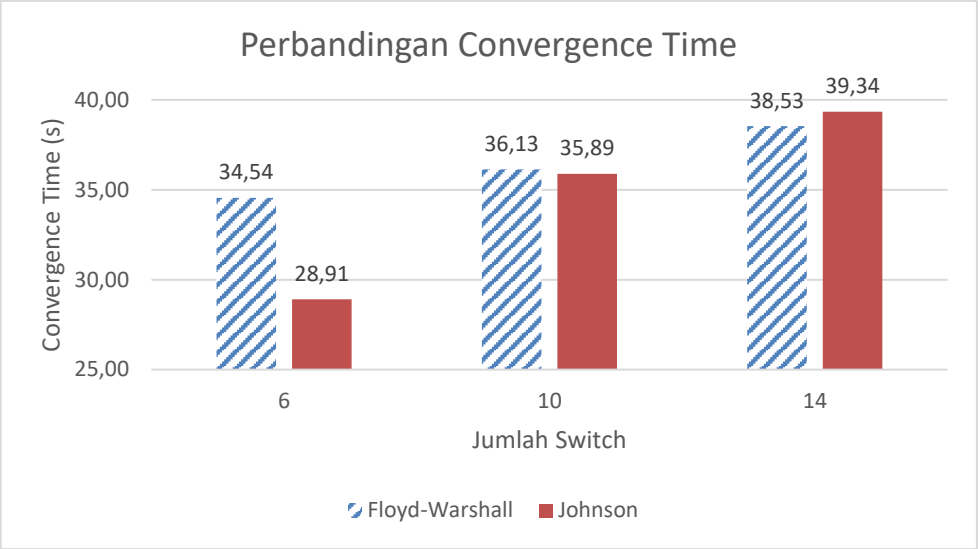
Untuk pengujian *delay* kedua algoritme didapatkan hasil yang sama baiknya dan tidak terjadi perbedaan yang signifikan. Baik Algoritme Floyd-Warshall maupun Algoritme Johnson diperoleh *delay* di bawah 10 ms untuk seluruh skenario. Perbandingan hasil pengujian *delay* dapat dilihat pada Gambar 5.9. Hasil *delay* yang diperoleh berada di bawah 10 ms karena setelah rute ditemukan oleh masing-masing algoritme tidak akan dilakukan proses pencarian rute lagi sehingga nilai *delay* yang diperoleh rendah. Kemudian mininmnya *congestion* yang terjadi juga berpengaruh pada nilai *delay* yang cenderung kecil.




**Gambar 5.9 Grafik Perbandingan *Delay***

### 5.2.4 Analisis Hasil Pengujian *Convergence Time*

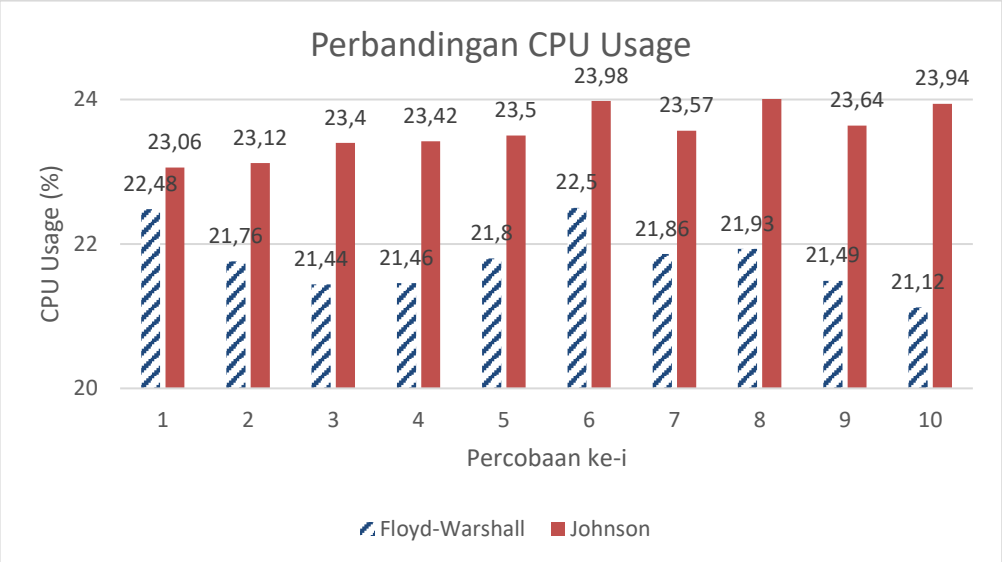
Berdasarkan Gambar 5.10 *convergence time* Algoritme Johnson lebih unggul dibanding Algoritme Floyd-Warshall pada topologi dengan jumlah *switch* 6 dan 10. Rata-rata *convergence time* yang diperoleh Algoritme Johnson pada 6 *switch* adalah 28,91 detik berbanding 34,54 detik waktu yang didapat Algoritme Floyd-Warshall. Sedangkan pada topologi dengan jumlah *switch* 14, Algoritme Floyd-Warshall sedikit lebih cepat dengan rata-rata *convergence time* 38,53 detik dibandingkan Algoritme Johnson dengan 39,34 detik. Berdasarkan hasil pengujian tersebut, Algoritme Johnson lebih unggul pada topologi dengan jumlah *node* 6 dan 10 karena karakteristiknya yang efektif pada topologi dengan *node* renggang/sedikit. Berkebalikan dengan Algoritme Johnson, Algoritme Floyd-Warshall karakteristiknya lebih efektif pada topologi dengan jumlah *switch* yang padat/banyak sehingga diperoleh hasil lebih baik pada saat *node* berjumlah 14 *switch*.




**Gambar 5.10 Grafik Perbandingan *Convergence Time***

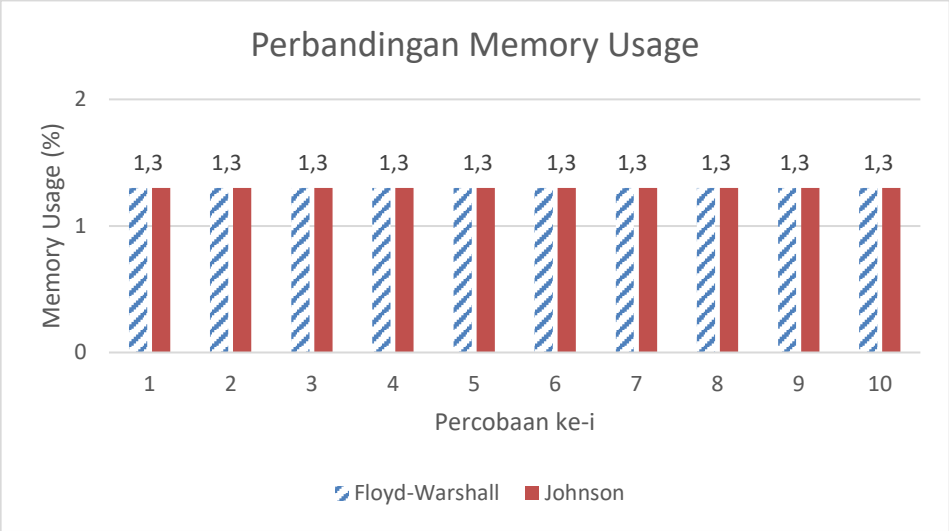
### 5.2.5 Analisis Hasil Pengujian CPU & *Memory Usage*

Berdasarkan Gambar 5.11, dari hasil pengujian yang telah dilakukan dapat diketahui bahwa Algoritme Floyd-Warshall lebih unggul dibandingkan Algoritme Johnson. Algoritme Johnson membutuhkan konsumsi CPU yang lebih besar daripada Algoritme Floyd-Warshall. Kompleksitas program Algoritme Johnson berpengaruh pada penggunaan resource. Pada Algoritme Johnson terdapat subrutin Bellman-Ford yang digunakan sekali untuk perhitungan bobot ulang pada setiap *link* dan Dijkstra yang dipanggil sebanyak jumlah *node* di topologi pada setiap kali proses Algoritme Johnson dalam menentukan rute terpendek. Sedangkan Algoritme Floyd-Warshall proses pencarian rutenya cukup sederhana sehingga lebih hemat dalam konsumsi daya CPU.




**Gambar 5.11 Grafik Perbandingan CPU Usage**

Sedangkan untuk pengujian *memory usage* untuk kedua algoritme tidak mengalami perubahan pada setiap skenario. Pada topologi berjumlah 10 *switch*, Algoritme Floyd-Warshall maupun Algoritme Johnson sama-sama mengonsumsi *memory usage* yang secara konsisten berada pada angka 1,3% seperti yang dapat dilihat pada Gambar 5.12. Baik Algoritme Johnson maupun Algoritme Floyd-Warshall tidak memerlukan resource yang besar karena *memory* hanya digunakan saat *controller* mulai dijalankan.




**Gambar 5.12 Grafik Perbandingan *Memory Usage***

# BAB 6 PENUTUP

Bab ini membahas kesimpulan yang diperoleh dari penelitian yang dilakukan yaitu tentang "Analisis Perbandingan Performansi Algoritme Floyd-Warshall dan Algoritme Johnson Untuk Penentuan Rute Terpendek Pada *Software Defined Network*" dan saran-saran untuk pengembangan topik skripsi arau pengembangan penelitian lebih lanjut.

## 6.1 Kesimpulan

Kesimpulan yang didapat berdasarkan pengujian dan 45nalisa yang telah dilakukan terhadap tugas akhir ini adalah sebagai berikut:

1. Implementasi Algoritme Floyd-Warshall dan Algoritme Johnson menggunakan simulator mininet dan *controller* ryu berjalan dengan baik sesuai yang diharapkan untuk melihat kinerjanya dalam menentukan rute terpendek.
2. Diperoleh hasil pengujian berdasarkan parameter uji:
   a. *Packet Loss*, pada pengujian Algoritme Floyd-Warshall diperoleh nilai rata-rata 2,33%, sedangkan Algoritme Johnson diperoleh nilai rata-rata 2,48%.
   b. *Delay*, pada pengujian Algoritme Floyd-Warshall diperoleh nilai rata-rata 4,08 ms, sedangkan Algoritme Johnson diperoleh nilai rata-rata 4,02 ms.
   c. *Convergence time*, pada pengujian Algoritme Floyd-Warshall diperoleh nilai rata-rata 34,54, 36,13 dan 38,53 detik dengan masing-masing topologi 6, 10 dan 14 *switch*, sedangkan Algoritme Johnson pada topologi 6, 10 dan 14 *switch* diperoleh masing-masing nilai rata-rata 29,91, 35,89 dan 39,94 detik.
   d. CPU usage, pada pengujian Algoritme Floyd-Warshall diperoleh rata-rata 21,78%, sedangkan Algoritme Johnson diperoleh nilai rata-rata 23,84%.
   e. *Memory usage* pada pengujian kedua algoritme sama-sama diperoleh nilai rata-rata 1,3% untuk seluruh percobaan.
3. Diperoleh analisa terhadap hasil pengujian:
   a. Masing-masing algoritme *routing* telah mampu untuk melakukan pencarian jalur terpendek pada topologi.
   b. Hasil pengujian *packet loss* tidak mengalami perbedaan yang signifikan, *packet loss* yang terjadi pada Algoritme Johnson dan Algoritme Floyd-Warshall rendah karena setelah rute ditemukan minim terjadi *collision* atau *congestion* yang merupakan faktor yang dapat mempengaruhi *packet loss*.
   c. Pada pengujian *delay*, kedua algoritme tidak memiliki perbedaan yang signifikan. Baik Algoritme Floyd-Warshall maupun Algoritme Johnson diperoleh hasil yang cenderung kecil karena setelah rute ditemukan tidak akan dilakukan proses pencarian rute lagi dan minim terjadinya *congestion*.
   d. Berdasarkan hasil pengujian *convergence time*, Algoritme Johnson didapatkan hasil yang lebih unggul pada topologi dengan jumlah *switch* 6 dan 10 sebab Algoritme Johnson efektif pada topologi dengan *node* yang

renggang. Di sisi lain Algoritme Floyd-Warshall efektif pada topologi dengan *node* yang padat, sehingga lebih cepat ketika *switch* berjumlah 14.

e. Pada pengujian CPU usage, Algoritme Johnson mengonsumsi resource lebih besar dari Algoritme Floyd-Warshall disebabkan pada proses penentuan rute terpendek, Algoritme Johnson melibatkan subrutin Bellman-Ford untuk perhitungan ulang bobot dan Dijkstra untuk mencari jalur sehingga membutuhkan resource yang lebih besar.
f. Hasil pengujian *memory usage* tidak mengalami perubahan pada setiap skenario, masing-masing algoritme mengonsumsi *memory* yang sama besar karena penggunaan *memory* hanya pada saat menjalankan *controller*.

## 6.2 Saran

Berdasarkan hasil dari pengujian sistem, dapat diberikan saran-saran untuk pengembangan selanjutnya, antara lain:

1. Penelitian selanjutnya dapat menggunakan algoritme *routing* lain untuk diimplementasikan pada *software defined network*.
2. Penelitian selanjutnya dapat menambahkan parameter uji yang lain agar kinerja algoritme dapat diketahui dan dianalisa lebih lanjut.
3. Penelitian selanjutnya dapat menggunakan *controller* lain sebagai perbandingan pada *control plane*.

# DAFTAR PUSTAKA


Azodolmolky, S., 2013. *Software Defined Network with OpenFlow.* Birmingham: Packt Publishing Ltd..

Brilliant, n.d. *Brilliant: Math and Science Done Right.* [Online] Available at: https://brilliant.org/wiki/johnsons-algorithm/ [Accessed 20 Februari 2017].

Ilhamsyah, M., 2016. Simulasi dan Uji Kinerja Algoritma johnson untuk Penentuan Rute Terbaik pada Jaringan Software Defined Network.

Iskandar, I., 2015. Analisa Quality of Service (QoS) Jaringan Internet Kampus. p. 10.

Kamayudi, A., 2006. Studi dan Implementasi Algoritma Dijkstra, Bellman-Ford dan Floyd-Warshall dalam Menangani Masalah Lintasan Terpendek dalam Graf. p. 6.

Kreutz, D. et al., 2014. Software-Defined Networking: A Comprehensive Survey. p. 63.

Kurose, J. F. & Ross, K. W., 2013. *Computer Networking: A Top-Down Approach.* 6 ed. New Jersey: Pearson Education, Inc..

McKeown, N. et al., 2008. OpenFlow: Enabling Innovation in Campus Networks. 38(2), p. 6.

Mininet, n.d. *Mininet: An Instant Virtual Network on your Laptop (or other PC).* [Online] Available at: http://mininet.org/ [Accessed 20 Februari 2017].

Nadeau, T. D. & Gray, K., 2013. *SDN: Software Defined Network.* Sebastopol: O'Reilly.

Open Networking Foundation, n.d. *OpenFlow - Open Networking Foundation.* [Online] Available at: https://www.opennetworking.org/sdn-resources/openflow [Accessed 23 Februari 2017].

Pandya, K., 2013. Network Structure or Topology. 1(2), p. 6.

Pranata, Y. A., 2016. ANALISIS OPTIMASI KINERJA QUALITY OF SERVICE PADA LAYANAN KOMUNIKASI DATA MENGGUNAKAN NS-2 DI PT. PLN (PERSERO) JEMBER. p. 8.

Ryu Development Team, 2017. *Ryu Documentation.* [Online] Available at: https://media.readthedocs.org/pdf/ryu-docs/latest/ryu-docs.pdf [Accessed 1 Maret 2017].

Saputra, I. A., 2016. Uji Performansi Algoritma Floyd-Warshall pada Jaringan Software Defined Network.

Techopedia, n.d. *Techopedia.* [Online]
Available at: https://www.techopedia.com/definition/28291/cpu-utilization
[Accessed 20 Juli 2017].

# LAMPIRAN

Lampiran 1 Hasil Pengujian *Packet Loss* 1

| Algoritme Johnson | | | | |
|---|---|---|---|---|
| No | Switch | | Parameter | |
| | Asal | Tujuan | Bandwidth (Mb) | *Packet Loss* (%) |
| 1 | 6 | 1 | 125 | 4,7 |
| 2 | 6 | 1 | 250 | 3,5 |
| 3 | 6 | 1 | 375 | 3,5 |
| 4 | 6 | 1 | 500 | 2,2 |
| 5 | 6 | 1 | 625 | 2,1 |
| 6 | 6 | 1 | 750 | 1,9 |
| 7 | 6 | 1 | 875 | 0,18 |
| 8 | 6 | 1 | 1000 | 0,026 |

| Algoritme Floyd-Warshall | | | | |
|---|---|---|---|---|
| No | Switch | | Parameter | |
| | Asal | Tujuan | Bandwidth (Mb) | *Packet Loss* (%) |
| 1 | 6 | 1 | 125 | 5 |
| 2 | 6 | 1 | 250 | 4,2 |
| 3 | 6 | 1 | 375 | 3,4 |
| 4 | 6 | 1 | 500 | 2,4 |
| 5 | 6 | 1 | 625 | 2,4 |
| 6 | 6 | 1 | 750 | 1,4 |
| 7 | 6 | 1 | 875 | 0,53 |
| 8 | 6 | 1 | 1000 | 0,074 |

Lampiran 2 Hasil Pengujian *Packet Loss* 2

| Algoritme Johnson | | | | |
|---|---|---|---|---|
| No | Switch | | Parameter | |
| | Asal | Tujuan | Bandwidth (Mb) | *Packet Loss* (%) |
| 1 | 5 | 2 | 125 | 5,8 |

| | | | | |
|---|---|---|---|---|
| 2 | 5 | 2 | 250 | 4,1 |
| 3 | 5 | 2 | 375 | 3,8 |
| 4 | 5 | 2 | 500 | 2,4 |
| 5 | 5 | 2 | 625 | 2 |
| 6 | 5 | 2 | 750 | 1,9 |
| 7 | 5 | 2 | 875 | 1,2 |
| 8 | 5 | 2 | 1000 | 0,048 |

| Algoritme Floyd-Warshall | | | | |
|---|---|---|---|---|
| No | Switch | | Parameter | |
| | Asal | Tujuan | Bandwidth (Mb) | *Packet Loss* (%) |
| 1 | 5 | 2 | 125 | 5,2 |
| 2 | 5 | 2 | 250 | 3,3 |
| 3 | 5 | 2 | 375 | 1,2 |
| 4 | 5 | 2 | 500 | 2,5 |
| 5 | 5 | 2 | 625 | 2,8 |
| 6 | 5 | 2 | 750 | 1,4 |
| 7 | 5 | 2 | 875 | 0,9 |
| 8 | 5 | 2 | 1000 | 0,078 |

Lampiran 3 Hasil Pengujian *Packet Loss* 3

| Algoritme Johnson | | | | |
|---|---|---|---|---|
| No | Switch | | Parameter | |
| | Asal | Tujuan | Bandwidth (Mb) | *Packet Loss* (%) |
| 1 | 4 | 3 | 125 | 6,5 |
| 2 | 4 | 3 | 250 | 3,3 |
| 3 | 4 | 3 | 375 | 3,4 |
| 4 | 4 | 3 | 500 | 2 |
| 5 | 4 | 3 | 625 | 2,8 |
| 6 | 4 | 3 | 750 | 1,3 |
| 7 | 4 | 3 | 875 | 0,68 |
| 8 | 4 | 3 | 1000 | 0,1 |

| Algoritme Floyd-Warshall | | | | |
|---|---|---|---|---|
| No | Switch | | Parameter | |
| | Asal | Tujuan | Bandwidth (Mb) | *Packet Loss* (%) |
| 1 | 4 | 3 | 125 | 4,9 |
| 2 | 4 | 3 | 250 | 4,8 |
| 3 | 4 | 3 | 375 | 2,3 |
| 4 | 4 | 3 | 500 | 2,6 |
| 5 | 4 | 3 | 625 | 2,2 |
| 6 | 4 | 3 | 750 | 1,6 |
| 7 | 4 | 3 | 875 | 0,6 |
| 8 | 4 | 3 | 1000 | 0,022 |

Lampiran 4 Hasil Pengujian *Delay* 1

| Algoritme Johnson | | | | |
|---|---|---|---|---|
| No | Switch | | Parameter | |
| | Asal | Tujuan | Ukuran Paket (byte) | *Delay* (ms) |
| 1 | 6 | 1 | 100 | 1,201 |
| 2 | 6 | 1 | 500 | 1,131 |
| 3 | 6 | 1 | 1000 | 1,185 |
| 4 | 6 | 1 | 2500 | 1,712 |
| 5 | 6 | 1 | 5000 | 2,571 |
| 6 | 6 | 1 | 10000 | 3,384 |
| 7 | 6 | 1 | 15000 | 5,504 |
| 8 | 6 | 1 | 20000 | 6,169 |
| 9 | 6 | 1 | 25000 | 7,633 |
| 10 | 6 | 1 | 30000 | 8,476 |

| Algoritme Floyd-Warshall | | | | |
|---|---|---|---|---|
| No | Switch | | Parameter | |
| | Asal | Tujuan | Ukuran Paket (byte) | *Delay* (ms) |
| 1 | 6 | 1 | 100 | 1,109 |

| | | | | |
|---|---|---|---|---|
| 2 | 6 | 1 | 500 | 1,169 |
| 3 | 6 | 1 | 1000 | 1,233 |
| 4 | 6 | 1 | 2500 | 1,596 |
| 5 | 6 | 1 | 5000 | 2,361 |
| 6 | 6 | 1 | 10000 | 3,533 |
| 7 | 6 | 1 | 15000 | 4,967 |
| 8 | 6 | 1 | 20000 | 6,347 |
| 9 | 6 | 1 | 25000 | 7,231 |
| 10 | 6 | 1 | 30000 | 9,037 |

Lampiran 5 Hasil Pengujian *Delay* 2

| Algoritme Johnson | | | | |
|---|---|---|---|---|
| No | Switch | | Parameter | |
| | Asal | Tujuan | Ukuran Paket (byte) | *Delay* (ms) |
| 1 | 5 | 2 | 100 | 0,979 |
| 2 | 5 | 2 | 500 | 1,248 |
| 3 | 5 | 2 | 1000 | 1,307 |
| 4 | 5 | 2 | 2500 | 1,727 |
| 5 | 5 | 2 | 5000 | 3,064 |
| 6 | 5 | 2 | 10000 | 2,669 |
| 7 | 5 | 2 | 15000 | 5,479 |
| 8 | 5 | 2 | 20000 | 5,52 |
| 9 | 5 | 2 | 25000 | 6,115 |
| 10 | 5 | 2 | 30000 | 7,614 |

| Algoritme Floyd-Warshall | | | | |
|---|---|---|---|---|
| No | Switch | | Parameter | |
| | Asal | Tujuan | Ukuran Paket (byte) | *Delay* (ms) |
| 1 | 5 | 2 | 100 | 1,215 |
| 2 | 5 | 2 | 500 | 1,545 |
| 3 | 5 | 2 | 1000 | 1,113 |
| 4 | 5 | 2 | 2500 | 1,637 |

| | | | | |
|---|---|---|---|---|
| 5 | 5 | 2 | 5000 | 3,219 |
| 6 | 5 | 2 | 10000 | 3,324 |
| 7 | 5 | 2 | 15000 | 4,634 |
| 8 | 5 | 2 | 20000 | 5,748 |
| 9 | 5 | 2 | 25000 | 7,103 |
| 10 | 5 | 2 | 30000 | 8,194 |

Lampiran 6 Hasil Pengujian *Delay* 3

| Algoritme Johnson | | | | |
|---|---|---|---|---|
| No | Switch | | Parameter | |
| | Asal | Tujuan | Ukuran Paket (byte) | *Delay* (ms) |
| 1 | 4 | 3 | 100 | 1,332 |
| 2 | 4 | 3 | 500 | 0,768 |
| 3 | 4 | 3 | 1000 | 1,212 |
| 4 | 4 | 3 | 2500 | 1,749 |
| 5 | 4 | 3 | 5000 | 2,807 |
| 6 | 4 | 3 | 10000 | 4,139 |
| 7 | 4 | 3 | 15000 | 6,664 |
| 8 | 4 | 3 | 20000 | 7,861 |
| 9 | 4 | 3 | 25000 | 8,644 |
| 10 | 4 | 3 | 30000 | 10,605 |

| Algoritme Floyd-Warshall | | | | |
|---|---|---|---|---|
| No | Switch | | Parameter | |
| | Asal | Tujuan | Ukuran Paket (byte) | *Delay* (ms) |
| 1 | 4 | 3 | 100 | 1,335 |
| 2 | 4 | 3 | 500 | 1,33 |
| 3 | 4 | 3 | 1000 | 1,494 |
| 4 | 4 | 3 | 2500 | 1,801 |
| 5 | 4 | 3 | 5000 | 2,485 |
| 6 | 4 | 3 | 10000 | 4,171 |
| 7 | 4 | 3 | 15000 | 6,141 |

| | | | | |
|---|---|---|---|---|
| 8 | 4 | 3 | 20000 | 7,386 |
| 9 | 4 | 3 | 25000 | 9,285 |
| 10 | 4 | 3 | 30000 | 10,75 |

Lampiran 7 Hasil Pengujian *Convergence Time*

| *Convergence Time* Algoritme Johnson | | | |
|---|---|---|---|
| No | Jumlah Switch | | |
| | 6 | 10 | 14 |
| 1 | 27,53 | 37,44 | 37,88 |
| 2 | 29,56 | 39,16 | 38,5 |
| 3 | 28,84 | 32,8 | 40,39 |
| 4 | 28,17 | 38,21 | 38,20 |
| 5 | 29,29 | 33,67 | 40,29 |
| 6 | 28,19 | 36,68 | 38,05 |
| 7 | 29,08 | 33,34 | 38,62 |
| 8 | 30,10 | 33,32 | 39,83 |
| 9 | 30,26 | 36,24 | 40,63 |
| 10 | 28,12 | 38,07 | 40,99 |

| *Convergence Time* Algoritme Floyd-Warshall | | | |
|---|---|---|---|
| No | Jumlah Switch | | |
| | 6 | 10 | 14 |
| 1 | 37,64 | 37,63 | 35,39 |
| 2 | 32,1 | 37,96 | 37,9 |
| 3 | 35,32 | 35,83 | 41,14 |
| 4 | 35,06 | 35,04 | 39,14 |
| 5 | 35,70 | 35,19 | 38,93 |
| 6 | 35,82 | 35,41 | 38,90 |
| 7 | 32,86 | 35,36 | 40,81 |
| 8 | 35,24 | 36,40 | 36,52 |
| 9 | 32,80 | 35,35 | 36,04 |
| 10 | 32,89 | 37,14 | 40,55 |