SKRIPSI

ANALISIS KINERJA PROTOKOL ROUTING OSPF MENGGUNAKAN CONTROLLER RYU DAN OPENDAYLIGHT PADA JARINGAN SOFTWARE DEFINED NETWORK (SDN)

Disusun dan diajukan oleh:

MUH. NAUFAL FALIQ
D121171503

PROGRAM STUDI SARJANA TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS HASANUDDIN
GOWA
2024

LEMBAR PENGESAHAN SKRIPSI

ANALISIS KINERJA PROTOKOL ROUTING OSPF MENGGUNAKAN CONTROLLER RYU DAN OPENDAYLIGHT PADA JARINGAN SOFTWARE DEFINED NETWORK (SDN)

Disusun dan diajukan oleh

Muhammad Naufal Faliq
D121171503

Telah dipertahankan di hadapan Panitia Ujian yang dibentuk dalam rangka Penyelesaian Studi Program Sarjana Program Studi Teknik Informatika
Fakultas Teknik Universitas Hasanuddin
Pada tanggal ..31 Juli 2024
dan dinyatakan telah memenuhi syarat kelulusan

Menyetujui,

Pembimbing Utama,	Pembimbing Pendamping,
Dr. Eng. Muhammad Niswar, S.T., M.IT	Dr. Eng. Zulkifli Tahir, S.T., M.Sc
NIP. 19730922 199903 1 001	NIP. 19840403 201012 1 004

Ketua Program Studi,

Prof. Dr. Ir. Indrabayu, S.T., M.T., M.Bus.Sys., IPM, ASEAN Eng.
NIP. 19750716 200212 1 004

PERNYATAAN KEASLIAN

Yang bertanda tangan dibawah ini ;
Nama : Muhammad Naufal Faliq
NIM : D121171503
Program Studi : Teknik Informatika
Jenjang : S1

Menyatakan dengan ini bahwa karya tulisan saya berjudul

Analisis Kinerja Protokol Routing OSPF Menggunakan Controller RYU dan OpenDaylight Pada Jaringan Software Defined Network (SDN)

Adalah karya tulisan saya sendiri dan bukan merupakan pengambilan alihan tulisan orang lain dan bahwa skripsi yang saya tulis ini benar-benar merupakan hasil karya saya sendiri.

Semua informasi yang ditulis dalam skripsi yang berasal dari penulis lain telah diberi penghargaan, yakni dengan mengutip sumber dan tahun penerbitnya. Oleh karena itu, semua tulisan dalam skripsi ini sepenuhnya menjadi tanggung jawab penulis. Apabila ada pihak manapun yang merasa ada kesamaan judul dan atau hasil temuan dalam skripsi ini, maka penulis siap untuk diklarifikasi dan mempertanggung jawabkan segala resiko.

Segala data dan informasi yang diperoleh selama proses pembuatan skripsi, yang akan dipublikasi oleh Penulis di masa depan harus mendapat persetujuan dari Dosen Pembimbing.

Apabila dikemudian hari terbukti atau dapat dibuktikan bahwa sebagian atau keselur
uhan isi skripsi ini hasil karya orang lain, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Gowa, 31 Juli 2024

Yang Menyatakan

Muhammad Naufal Faliq
NIM. D121 17 1503

ABSTRAK

MUHAMMAD NAUFAL FALIQ. *Analisis Kinerja Protokol Routing OSPF Menggunakan Controller RYU dan OpenDaylight Pada Jaringan Software Defined Network (SDN)* (dibimbing oleh Dr. Eng. Muhammad Niswar, S.T., M.IT dan Dr. Eng. Zulkifli Tahir, S.T., M.Sc)

Jaringan *Software Defined Network* (SDN) adalah paradigma jaringan yang mengubah cara tradisional dalam mengelola dan mengontrol jaringan. Salah satu komponen kunci dalam SDN adalah penggunaan controller yang memfasilitasi pengaturan dinamis dan sentralisasi manajemen jaringan. Protokol routing OSPF (*Open Shortest Path First*) telah lama digunakan dalam jaringan tradisional untuk mengatur lalu lintas data dengan efisien. Penelitian ini bertujuan untuk menganalisis kinerja protokol OSPF dalam konteks SDN dengan menggunakan controller RYU dan OpenDaylight.

Metode penelitian yang digunakan meliputi pengaturan simulasi dengan topologi jaringan yang disimulasikan menggunakan perangkat lunak Mininet. Pengujian dilakukan dengan memvariasikan jumlah node dan karakteristik trafik untuk mengevaluasi kinerja protokol OSPF dalam hal *throughput, latency,* dan *packet loss*.

Kesimpulan dari penelitian ini adalah bahwa OSPF tetap menjadi pilihan yang solid untuk digunakan dalam lingkungan SDN, walaupun terdapat perbedan kinerja kedua controller. Dengan kemampuan adaptasi yang baik terhadap perubahan topologi jaringan yang dinamis. Saran untuk penelitian mendatang termasuk pengembangan strategi pengoptimalan OSPF yang lebih baik untuk SDN serta peningkatan keamanan dalam implementasi SDN yang melibatkan controller. Studi ini diharapkan dapat memberikan kontribusi yang berarti bagi pengembangan lebih lanjut dalam mengintegrasikan OSPF dengan SDN secara efektif.

Kata kunci : Software Defined Network (SDN), OSPF, RYU, OpenDaylight, routing, kinerja jaringan

ABSTRACT

MUHAMMAD NAUFAL FALIQ. *Performance Analysis of OSPF Routing Protocol Using RYU and OpenDaylight Controllers on Software Defined Networks (SDN)* (supervised by Dr. Eng. Muhammad Niswar, S.T., M.IT dan Dr. Eng. Zulkifli Tahir, S.T., M.Sc).

Software Defined Network (SDN) is a networking paradigm that changes the traditional way of managing and controlling networks. One of the key components in SDN is the use of controllers that facilitate dynamic settings and centralized network management. The OSPF (Open Shortest Path First) routing protocol has long been used in traditional networks to manage data traffic efficiently. This research aims to analyze the performance of the OSPF protocol in the SDN context using the RYU and OpenDaylight controllers.

The research method used includes a simulation setup with a simulated network topology using Mininet software. Testing was carried out by varying the number of nodes and traffic characteristics to evaluate the performance of the OSPF protocol in terms of throughput, latency, and packet loss.

The conclusion of this research is that OSPF remains a solid choice for use in an SDN environment, even though there are differences in the performance of the two controllers. With good adaptability to dynamic network topology changes. Suggestions for future research include developing better OSPF optimization strategies for SDN as well as improving security in SDN implementations involving controllers. This study is expected to provide a significant contribution to further development in effectively integrating OSPF with SDN.

Keywords : Software Defined Network (SDN), OSPF, RYU, OpenDaylight, routing, network performance

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR SINGKATAN DAN ARTI SIMBOL

Lambang/Singkatan	Arti dan Keterangan
SDN	*Software Defined Network*
OSPF	*Open Shortest Path First*
QoS	*Quality of Service*
ODL	*Opendaylight*
LAN	*Local Area Network*
WAN	*Wide Area Network*
APIs	*Aplication Protocol Interfaces*
ONOS	*Open Network Operating System*
NTT	*Nippon Telegraph dan Telephone*
ICMP	*Internet Control Message Protocol*
TTL	*Time-to-Live*
IP	*Internet Protocol*
IGP	*Interior Gateway routing Potokol*
ISP	*Internet Service Provider*
LSAs	*Link-State Advertisements*
LSDB	*Link-State Database*
DoS	*Denial of service*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
HTTP	*Hypertext Transfer Protocol*
DNS	*Domain Name System*
ISP	*Internet Service Provider*
RIP	*Routing Information Protocol*
BGP	*Border Gateway Protocol*
IS-IS	*Intermediate System to Intermediate System*

DAFTAR LAMPIRAN

KATA PENGANTAR

Puji syukur kita panjatkan kepada Allah SWT yang telah memberikan limpahan kenikmatan dan kesempatan dalam menyelesaikan tugas akhir ini. Salawat serta salam kita panjatkan pada Nabi Muhammad SAW sebagai suri tauladan, pelita dalam kegelapan hingga akhir zaman sehingga penulis dapat menyelesaikan penulisan skripsi yang berjudul **"*Analisis Kinerja Protokol Routing OSPF Menggunakan Controller RYU dan OpenDaylight Pada Jaringan Software Defined Network (SDN)*"** guna memenuhi salah satu persyaratan dalam menyelesaikan jenjang Strata-1 di Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin.

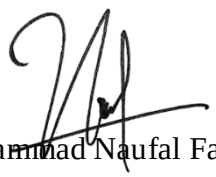
Penulis menyadari sepenuhnya bahwa skripsi ini masih jauh dari kesempurnaan karena menyadari segala keterbatasan yang ada. Dalam penulisan skripsi ini penulis menghadapi berbagai kendala dan masalah, namun itu tidak menurunkan semangat penulis dengan usaha yang maksimal dan kemampuan yang Allah berikan kepada penulis serta bantuan dan dukungan dari berbagai pihak, maka penulisan skripsi ini dapat selesai. Oleh karena itu, pada kesempatan ini penulis ingin menyampaikan ucapan terima kasih kepada :

1. Allah SWT, sebagai zat yang Maha segalanya sebagai penolong di setiap langkah hidup penulis,
2. Prof. Dr. Eng. Ir. Muhammad Isran Ramli, S.T., M.T., IPM., ASEAN Eng, selaku Dekan Fakultas Teknik Universitas Hasanuddin yang telah memberikan izin kepada penulis untuk melakukan penelitian,
3. Bapak Prof. Dr. Ir. Indrabayu, S.T., M.T., M.Bus.Sys., IPM, ASEAN. Eng. selaku Ketua Ketua Program Studi Departemen Informatika Fakultas Teknik Universitas Hasanuddin atas bantuannya dan bimbingannya kepada penulis,
4. Bapak Dr. Eng, Muhammad Niswar S.T., M.IT. selaku pembimbing utama dan Bapak Dr. Eng. Zulkifli Tahir, S.T., M.Sc selaku pembimbing pendamping yang senantiasa menyediakan waktu, tenaga, pikiran, dan perhatian yang luar biasa dalam mengarahkan penulis dalam penyusunan tugas akhir ini,

5. Bapak Robert, Bapak Zainuddin dan Ibu Yuanita serta segenap staf Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin yang telah membantu kelancaran penyelesaian tugas akhir penulis,
6. Segenap Dosen dan Staf Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin yang telah banyak membantu semasa perkuliahan hingga penyelesaian tugas akhir penulis,
7. Alm Ayahanda Herman MP, Ibunda Lenni Marlina, Kakak Heli Kaishelmi Herman dan Adek Reyhan Faqih Ashuri yang tercinta atas segala bantuan, bimbingan, dorongan serta doa restu yang diberikan kepada penulis selama penyusunan skripsi.
8. Ir. H. Muhammad Ilham Musa atas segala bantuan, bimbingan, dorongan serta doa restu yang diberikan kepada penulis selama penyusunan skripsi.
9. Seluruh keluarga besar saya yang tidak bisa saya sebutkan satu persatu yang juga telah memberikan banyak sumbangsih baik secara materi maupun non-materi kepada penulis,
10. Anak Teknik, baik junior maupun senior dan terkhusus untuk teman-teman angkatan 2017 yang menemani penulis dalam mengarungi suka dan duka dalam menjalani dunia perkuliahan dan kemahasiswaan di Fakultas Teknik Universitas Hasanuddin,
11. Saudara seperjuangan penulis RECOGN17ER yang telah menemani dan mendukung perjalanan penulis sekaligus tempat berbagi keluh kesah selama menjadi mahasiswa teknik di Departemen Informatika Fakultas Teknik Universitas Hasanuddin,
12. Seluruh pihak yang tidak sempat disebutkan satu persatu yang telah banyak meluangkan tenaga, waktu, dan pikiran selama penyusunan tugas akhir ini.

Makassar, 31 Juli 2024

Penulis,



Muhammad Naufal Faliq

BAB I
PENDAHULUAN

1.1 Latar Belakang

Penggunaan jaringan komputer saat ini semakin kompleks dengan jumlah perangkat yang semakin banyak. Untuk mengatasi kompleksitasi ini, penggunaan *Software Defined Network (SDN)* menjadi solusi yang efektif. SDN memungkinkan administrator jaringan untuk mengelola jaringan dengan lebih mudah dan efisien, serta mempercepat implementasi perubahan pada jaringan. SDN merupakan sebuah konsep baru pada pendekatan dalam merancang, menyusun serta mengelola jaringan komputer. Konsep ini berkenaan dengan arsitektur perangkat jaringan seperti router, packet switch, switch LAN dan lain sebagainya. Sebuah perangkat jaringan memiliki dua bagian yaitu data plane dan control plane. Konsep dari SDN ini menerapkan pemisahan antara data plane dan control plane, yang mana data plane tetap berada pada perangkat jaringan, sedangkan control plane terdapat dalam sebuah entity terpisah yang dinamakan sebagai controller. (Dwi Rahmawan & Risqiwati, 2020)

Pada jaringan *Software Defined Network (SDN)* ada beberapa protocol, diantaranya Openflow. Openflow merupakan protocol yang memungkinkan server memberitahukan switch jaringan tempat mengirim paket yang dimana berfungsi sebagai penghubung antara controller yaitu termasuk dalam control plane dengan data plane melewati secure channel lalu ke flow tabel dan diteruskan ke user. (홍종욱, 2019)

Protokol Openflow dapat diimplementasikan di berbagai platform, salah satunya RYU dan OpenDaylight. Kedua platform ini memiliki kelebihan dan kekurangan masing – masing. Selain itu, kedua controller tersebut akan diimplementasikan pada protocol routing OSPF. Protokol OSPF merupakan protokol routing yang lebih kompleks dan dapat menghasilkan jalur terpendek pada jaringan. Penggunaan OSPF pada jaringan SDN akan memberikan keuntungan dalam mengatur jalur yang optimal pada jaringan. Teknologi SDN yang terus berkembang seiring dengan kebutuhan akan jaringan terutama dalam melakukan konfigurasi jaringan yang semakin kompleks maka performansi menjadi hal krusial

yang harus diperhatikan. Tolak ukur pengukuran performansi pada jaringan dikenal dengan *Quality of Service (QoS).* Penelitian ini dilakukan untuk mengetahui bagaimana analisi kinerja dari protocol routing OSPF menggunakan controller RYU dan OpenDaylight pada topologi jaringan tree berdasarkan parameter QoS. (Iryani et al., 2021)

Oleh karena itu, penelitian ini akan membahas tentang "Analisis Kinerja Protokol Routing OSPF Menggunakan Controller RYU dan Opendaylight Pada Jaringan Software Defined Network (SDN)". Penelitian ini diharapkan dapat memberikan informasi tentang perbedaan kinerja antara kedua platform controller pada jaringan SDN dengan menggunakan protokol OSPF. Informasi ini dapat membantu administrator jaringan dalam memilih platform controller yang sesuai untuk jaringan SDN yang sedang dikelola.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang diuraikan, maka rumusan masalah pada penelitian ini antara lain :

1. Bagaimana implementasi routing protokol OSPF pada jaringan SDN yang menggunakan controller RYU dan OpenDaylight ?
2. Bagaimana performa jaringan SDN yang menggunakan controller RYU dan OpenDaylight dengan menggunakan routing protokol OSPF ?
3. Manakah controller yang lebih optimal untuk digunakan pada jaringan SDN dengan menggunakan routing protokol OSPF, RYU atau Opendaylight ?

1.3 Tujuan Penelitian/Perancangan

Tujuan akhir dari penelitian ini yakni :

1. Mengimplementasikan routing protokol OSPF pada jaringan SDN yang menggunakan controller RYU dan OpenDaylight.
2. Menganalisis performa jaringan SDN yang menggunakan controller RYU dan OpenDaylight dengan menggunakan routing protokol OSPF.
3. Menentukan controller yang lebih optimal untuk digunakan pada jaringan SDN dengan menggunakan routing protokol OSPF, apakah RYU atau OpenDaylight.

1.4 Manfaat Penelitian/Perancangan

Manfaat yang akan didapat pada penelitian ini, yakni :

1. Memberikan pemahaman yang lebih baik tentang karakteristik controller RYU dan OpenDaylight pada jaringan SDN serta bagaimana implementasi routing protokol OSPF pada kedua controller tersebut.
2. Membantu administrator jaringan dalam memilih controller yang lebih optimal untuk digunakan pada jaringan SDN dengan menggunakan routing protokol OSPF, sehinggan dapat meningkatkan performa jaringan.
3. Memberikan informasi yang berguna bagi pengembangan teknologi jaringan SDN dan meningkatkan literatur penelitian di bidang jaringan komputer.
4. Sebagai referensi bagi peneliti lain yang ingin melakukan penelitian serupa mengenai perbandingan antara controller RYU dan OpenDaylight pada jaringan SDN dengan menggunakan routing protokol OSPF.
5. Meningkatkan kemampuan dan keterampilan penulis dalam melakukan penelitian dan mengembangkan teknologi jaringan SDN.

1.5 Ruang Lingkup/Asumsi perancangan

Berdasarkan beberapa hal yang dicantumkan pada rumusan masalah, maka batasan dari penelitian ini yakni :

1. Penelitian ini akan berfokus pada perbandingan antara controller RYU dan OpenDaylight pada jaringan SDN dengan menggunakan routing protokol OSPF.
2. Penelitian ini hanya akan membahas performa jaringan pada aspek routing protokol OSPF dan tidak membahas aspek lain seperti keamanan, manajemen, dan monitoring jaringan.
3. Menggunakan topologi jaringan tree yang terdiri dari beberapa switch, router dan host.
4. Penelitian ini tidak akan membahas mengenai implementasi jaringan SDN secara keseluruhan, namun hanya akan membahas aspek yang berkaitan dengan perbandingan controller RYU dan OpenDaylight dengan menggunakan routing protokol OSPF.
5. Penelitian ini hanya akan dilakukan di lingkungan simulasi menggunakan perangkat lunak Mininet.

BAB II
TINJAUAN PUSTAKA

2.1 Jaringan Komputer

Jaringan komputer adalah kumpulan komputer atau perangkat komunikasi yang terhubung satu sama lain untuk berbagi sumber daya, seperti data, aplikasi, dan layanan. Tujuan dari jaringan komputer adalah memungkinkan komunikasi dan pertukaran informasi antara berbagai perangkat dalam suatu sistem. Jaringan komputer dapat memiliki cakupan yang bervariasi, mulai dari jaringan lokal (*Local Area Network* atau LAN) yang terbatas pada area geografis kecil, hingga jaringan luas (*Wide Area Network* atau WAN) yang dapat mencakup area geografis yang lebih besar, bahkan mencakup wilayah global.

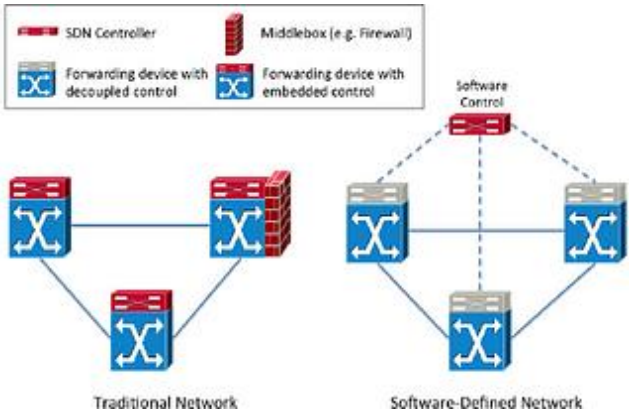
2.2 Software Defined Network (SDN)

Awal mula terciptanya teknologi *Software Defined Networking* dimulai tidak lama setelah Sun Mycrosystems merilis Java pada tahun 1995, namun pada saat itu belum cukup membangunkan para periset untuk mengembangkan teknologi tersebut. Baru pada tahun 2008 *Software Defined Network* ini dikembangkan di UC Barkeley and Stanford University. Dan kemudian teknologi tersebut mulai di promosikan oleh Open Networking Foundation yang didirikan pada tahun 2011 untuk memperkenalkan teknologi SDN dan OpenFlow. (Abidin, 2021)

2.2.1 Pengertian Software Defined Network (SDN)

Jaringan tradisional dengan arsitektur yang kaku dan kompleksitas yang tinggi sering kali tidak mampu memenuhi tuntutan modern ini. Di sinilah konsep *Software Defined Network (SDN)* hadir sebagai solusi inovatif SDN memungkinkan pemisahan antara lapisan kontrol dan lapisan data dalam jaringan, memberikan kemampuan untuk mengelola jaringan secara dinamis melalui perangkat lunak yang terpusat.

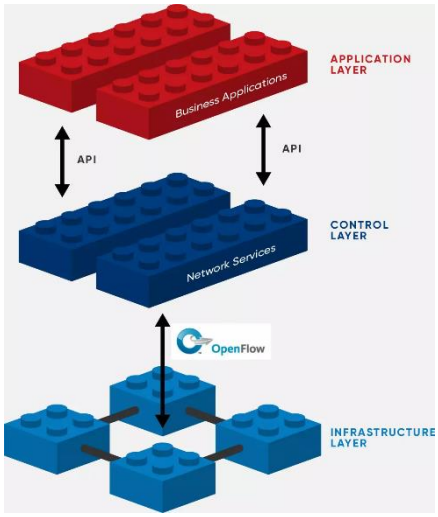
Software Defined Network (SDN) diusulkan untuk mengatasi kelemahan jaringan tradisional. Sebagai paradigma jaringan baru, SDN merevolusi teknologi jaringan dengan mematahkan ide dasar jaringan tradisional.



Gambar 2. 1 Perbedaan (a) Jaringan Tradisional dan (b) Jaringan SDN

Perbedaan mendasar antara jaringan tradisional dengan *Software Defined Network (SDN)* adalah penempatan fungsi control dan fordward. Pada jaringan tradisional fungsi control dan fordward ditempatkan pada device yang sama yaitu router. Sedangkan pada jaringan SDN fungsi kontrol ditempatkan pada software terpusat (controller) dan fungsi fordward ditempatkan pada suatu perangkat kosong berupa switch (fordwarding device). (Sulfiana, 2019)

SDN merupakan sebuah arsitektur jaringan yang sedang berkembang yang memisahkan antara data plane dan control plane dalam perangkat jaringan yang kemudian membuat control plane menjadi independent dan dapat diprogram. Pemisahan yang dilakukan antara data plane dan control plane menjadikannya abstraksi dan infrastruktur jaringan dan aplikasi yang membuat jaringan menjadi sebuah keberadaan virtual.



Gambar 2. 2 Arsitektur SDN

2.2.2 Konsep Software Defined Network (SDN)

Konsep *Software Defined Network (SDN)*, pertama kali diperkenalkan oleh Martin Casado di Universitas Stanford pada tahun 2007 dengan tulisan pada jurnalnya berjudul "Ethane : Talking Control of the Enterprise". (Thomas et al., 2018). Konsep dasar dari SDN memisahkan fungsi kontrol (kontrol plane) dan pengalihan lalu lintas (data plane). Fungsi kontrol ditempatkan di pusat, sementara perangkat keras jaringan (switches, router) bertanggung jawab untuk pengalihan lalu lintas sesuai dengan intruksi yang diberikan oleh kontroler. Kontroler pusat berperan sebagai otak jaringan. Ia mengambil keputusan terkait routing dan manajemen jaringan. Keputusan ini kemudian dikirimkan ke perangkat keras jaringan untuk diimplementasikan. Dalam SDN terdapat protocol yang menonjol yaitu Openflow. SDN menggunakan protocol komunikasi Openflow antara kontroler untuk mengirim intruksi kepada perangkat keras jaringan dan mengatur aliran lalu lintas.

Dari segi programmability, SDN memberikan kemampuan untuk mengelola jaringan secara programatik. Administrator dapat menggunakan antarmuka pemrograman untuk mengatur aturan, kebijakan, dan konfigurasi jaringan. Dengan kontrol yang terpusat dan kemampuan programatik, SDN memberikan kecepatan dan fleksibilitas yang tinggi dalam menyesuaikan jaringan dengan perubahan kebutuhan dan kondisi.

SDN memanfaatkan konsep virtualisasi untuk menciptakan jaringan virtual yang terisolasi. Hal ini memungkinkan penggunaan yang lebih efisien dari sumber daya jaringan dan mempermudah manajemen berbagai layanan. SDN mendukung orkestrasi, yang mencakup otomatisasi konfigurasi dan manajemen jaringan. Hal ini memungkinkan jaringan beradaptasi secara otomatis terhadap perubahan kondisi atau permintaan. Dalam hal ini, SDN memungkinkan pengelolaan jaringan berbasis kebijakan, dimana administrator dapat menentukan kebijakan secara terpusat dan mengimplementasikannya secara konsisten di selurh jaringan.

Oleh karena itu, konsep – konsep ini mengarah pada paradigma SDN yang memberikan pengelolaan yang lebih sentral, responsif, dan mudah diatur dalam konteks jaringan. SDN memberikan fleksibilitas dan skalabilitas yang lebih tinggi,

menghadirkan solusi yang dapat disesuaikan dengan kebutuhan bisnis yang berubah – ubah.

2.2.3 Tujuan Utama Software Defined Network (SDN)

Tujuan utama dari SDN untuk mencapai pengelolaan jaringan yang lebih baik dengan tingkatan dan kompleksitas yang besar serta memastikan bahwa semua keputusan dari sistem kontrol dibuat dari titik pusat (kontroller). SDN memperkenalkan suatu metode untuk meningkatkan tingkat abstraksi pada konfigurasi jaringan, menyediakan mekanisme yang secara otomatis bereaksi terhadap perubahan yang sering terjadi dan terus menerus untuk jaringan. SDN bertujuan untuk menyentralisasi kontrol jaringan, memungkinkan pengambilan keputusan yang lebih pintar dan efisien untuk routing dan manajemen sumber daya. SDN memberikan fleksibilitas yang tinggi dengan kemampuan penyesuaian jaringan secara dinamis, memungkinkan skalabilitas yang lebih baik sesuai dengan kebutuhan bisnis. SDN bertujuan untuk mengoptimalkan penggunaan sumber daya jaringan dan memberikan platform yang efisien untuk mengelola dan mengonfigurasi perangkat keras jaringan. SDN berupaya meningkatkan kontrol keamanan dan memberikan pengelolaan jaringan yang lebih efisien melalui perangkat lunak. SDN dirancang untuk mendukung pengembangan aplikasi dan inovasi dalam penggunaan sumber daya jaringan, sehingga merangsang perkembangan teknologi, sehinggan SDN mempermudah penerapan berbagai jenis macam aplikasi jaringan yang dapat diprogram pada controllernya, karena sifat controller SDN yang programmable tersebut. (Attamimi et al., 2017)

2.2.4 Karakteristik Software Defined Network (SDN)

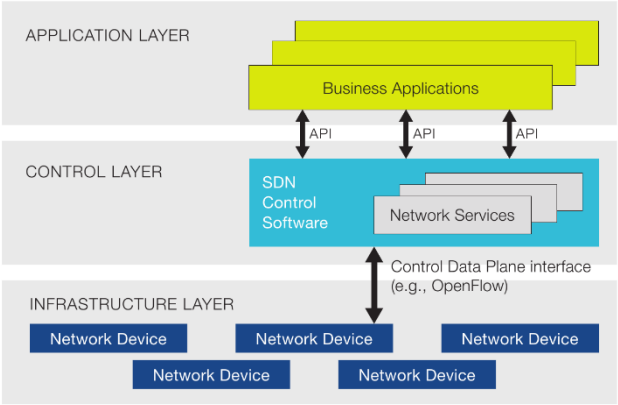
Software Defined Network (SDN) memiliki beberapa karakteristik kunci yang membedakannya dari pendekatan tradisional dalam pengelolaan jaringan, diantaranya :

1. SDN memisahkan fungsi kontrol plane, yang mengelola keputusan perutean dari data plane yang mengirimkan paket data. Ini memberikan fleksibilitas lebih besar dalam pengelolaan dan pengaturan jaringan.
2. SDN memungkinkan pengaturan dan konfigurasi jaringan secara dinamis melalui perangkat lunak.

3. SDN memberikan kontrol jaringan yang lebih sentral melalui pusat kontrol. Ini memungkinkan pengambilan keputusan jaringan yang lebih terpusat dan koheren.
4. SDN mendukung virtualisasi sumber daya jaringan, memungkinkan pembagian dan isolasi sumber daya secara efisien untuk mendukung multiple tenants atau layanan.
5. SDN sering kali didukung oleh standar terbuka dan protokol yang dapat diakses, memungkinkan interoperabilitas antara berbagai vendor dan perangkat.
6. SDN menyediakan antarmuka yang dapat diprogram (API) yang memungkinkan aplikasi dan perangkat lunak manajemen jaringan untuk berkomunikasi dengan perangkat keras jaringan.
7. SDN mendukung otomatisasi konfigurasi jaringan. Perangkat lunak kontrol dapat secara otomatis merespon perubahan kondisi jaringan dan menyesuaikan konfigurasi sesuai kebutuhan.
8. SDN memungkinkan jaringan untuk beradaptasi dengan perubahan secara dinamis, memungkinkan penyesuaian cepat terhadap permintaan dan kondisi jaringan yang berubah.
9. SDN dirancang untuk skalabilitas yang baik, memungkinkan pertumbuhan jaringan tanpa mengorbankan kinerja atau menghasilkan kompleksitas yang tidak terkelola.
10. SDN memungkinkan pengumpulan data analitik jaringan yang mendalam yang dapat digunakan untuk pemantauan, analisis dan pengambilan keputusan yang lebih terkait performa jaringan.

2.2.5 Arsitektur Software Defined Network (SDN)

Pada arsitektur *Software Defined Network (SDN)* setiap layer dapat bekerja secara independent dan berkomunikasi melalui antarmuka jaringan untuk memberikan fungsi berlapis dari perangkat fisik yang berbeda. Aspek arsitektur ini memungkinkan administrator jaringan untuk mengatasi beberapa tantangan dalam dunia jaringan komputer (Adrian, 2017).



Gambar 2. 3 Arsitektur Software Defined Network

Gambaran logis arsitektur Software Defined Network pada gambar 2.3 menunjukan terdapat 3 layer pada arsitektur *Software Defined Network (SDN)*, yaitu :

1. Aplication Layer

   Aplication layer merupakan layer yang berperan sebagai antar muka untuk memudahkan pengelola jaringan dalam melakukan fungsi konfigurasi, fungsi kontrol dan fungsi evaluasi. Pada lapisan ini terbentuk dari intergrasi dengan kontrol layer yang memberikan informasi tentang jaringan yang selanjutnya ditampilkan dalam bentuk yang dapat dipahami oleh pengelola jaringan (Irmawati et al., 2017).

2. Control Layer

   Lapisan kontrol berfungsi sebagai otak dari arsitektur SDN. Disini, perangkat lunak kontrol SDN berada bertugas mengambil keputusan terkait manajemen jaringan. Perangkat lunak kontrol inilah yang memahami topologi jaringan, kondisi, dan kebutuhan aplikasi. Contoh teknologi di lapisan kontrol termasuk controller SDN seperti OpenDaylight, ONOS, atau RYU.

3. Infrastructure Layer

   Lapisan infrastruktur atau data plane merupakan bagian fisik jaringan yang terdiri dari perangkat keras seperti switch dan router. Perangkat keras ini bertanggung jawab untuk mengirimkan paket data sesuai dengan intruksi

yang diberikan oleh lapisan kontrol. Pada lapisan ini, terdapat perangkat jaringan yang dapat diatur dan dikonfigurasi oleh perangkat lunak kontrol.

2.2.6 Keunggulan Arsitektur Software Defined Network (SDN)

Software Defined Network (SDN) memiliki beberapa keunggulan yang membuatnya menjadi pendekatan yang menarik dalam pengelolaan jaringan. Keungguan SDN antara lain :

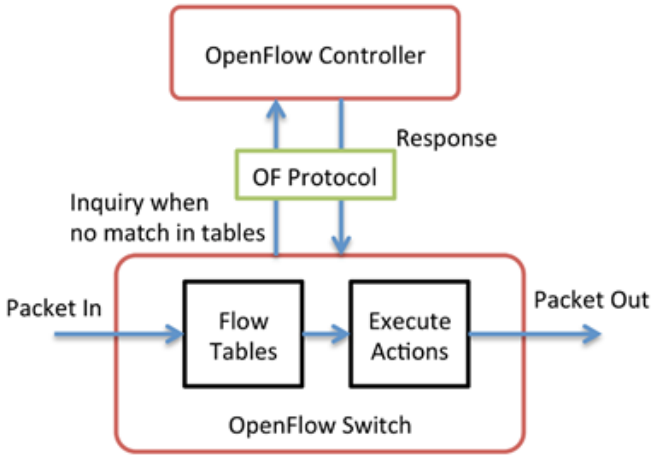
1. Dapat diprogramkan secara langsung karena kontrol panelnya terpisah dari *forwarding plane* (data plane).
2. Pemisahan antara kontrol plane dan data plane menyebabkan administrator jaringan dapat secara dinamis menyesuaikan *flow traffic network* sesuai dengan kebutuhan instusi.
3. *Network intelligence* mengelola satu gambaran umum yang utuh mengenai jaringan.
4. SDN memungkinkan jaringan administrator untuk mengelola sumber daya jaringan dengan sangat cepat.
5. Controller plane tidak bergantung pada *vendor-specifik device* atau protokol tertentu.
6. Penyajian antar muka yang lebih baik sebagai aplikasi manajemen jaringan terpusat.

2.3 Protokol Openflow

Jadi awal mula dari Openflow merupakan protokol terbilang relatif baru yang dirancang dan diimplementasikan di Stanford University pada tahun 2008. Sejak dimulainya standarisasi Protokol Openflow telah berkembang. Protokol Openflow adalah salah satu jenis dari APIs (*Aplication Protocol Interfaces*) dalam jaringan SDN yang digunakan untuk mengontrol dan mengatur *traffic flows* pada switch dalam sebuah jaringan, jadi singkatnya kontrol plane berkomunikasi dengan data plane melalui Openflow. Openflow dapat bekerja pada switch dari berbagai vendor.

Openflow adalah protokol paling utama pada SDN. Openflow terletak diantara kontrol plane dan data plane. Openflow mengatur routing dan pengiriman paket ketika melalui switch. Dalam sebuah jaringan, switch berfungsi meneruskan paket yang melalui suatu port tanpa mampu membedakan tipe protokol data yang dikirimkan. Openflow memungkinkan untuk mengakses dan mampu memanipulasi data plane secara langsung dari perangkat jaringan seperti switch dan router baik secara fisik maupun virtual (Ummah, 2016).

Agar Openflow dapat bekerja sesuai tujuan dan fungsinya, Openflow mempunyai 2 komponen penting yaitu Openflow Controller dan Openflow Switch.



Gambar 2. 4 Openflow Controller dan Openflow Switch

2.3.1 Openflow Controller

Openflow controller bertugas mengontrol path, memformulasikan flow dan mengatur kerja dari Openflow switch. Terdapat beberapa Openflow controller yang dapat digunakan seperti NOX (C base), POX (phyton base), dan Floodlight (java base).

2.3.2 Openflow Switch

Pada perangkat Openflow switch sendiri dapat dibagi menjadi tiga bagian yaitu :

1. Flow Table

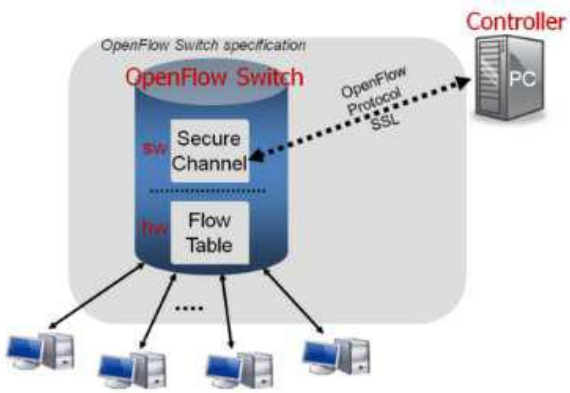
   Sebuah *flow table* yang mengindikasikan bahwa switch harus memroses flow yang ada di dalamnya. Daftar flow ini dibuat berdasarkan *actions* yang mana bersinggungan langsung dengan setiap flow.

2. Secured Chanel

   Sebuah saluran yang aman dibutuhkan untuk menghubungkan switch dengan controller. Melalui saluran ini, Openflow menyediakan jalur komunikasi antara switch dan controller melalui protokol yang disebut protocol Openflow.

3. Protokol Openflow

   Protokol ini menyediakan sebuah standar dan komunikasi terbuka antara controller dan switch. Openflow controller menentukan ke interface manakah flow akan diterapkan dari flow table (Khoerul & Ronald, 2017).



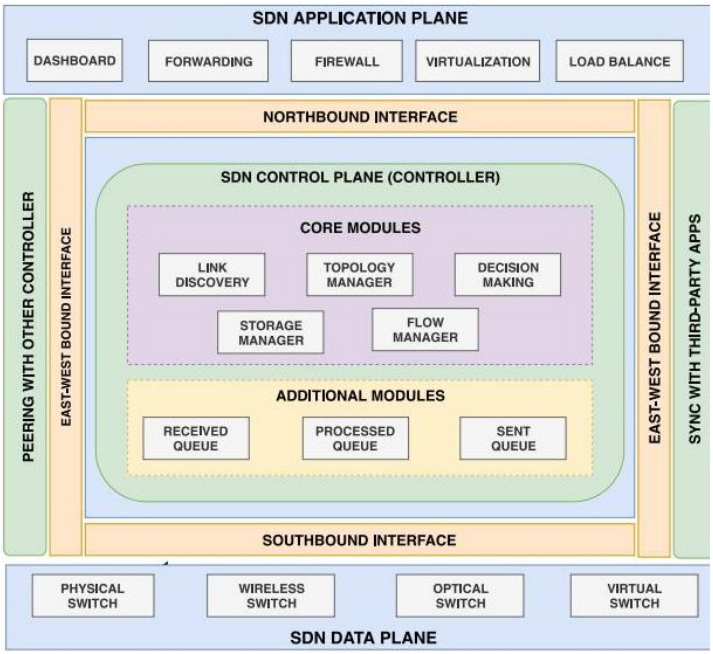
Gambar 2. 5 Mekanisme Switch Openflow

2.4 Controller Software Defined Network (SDN)

Controller SDN adalah aplikasi SDN yang mengelola flow kontrol untuk mengaktifkan *intelligence networking*. Controller SDN bekerja berdasarkan protokol seperti Openflow yang memungkinkan server memberitahu kemana paket dikirimkan. Perangkat meneruskan paket data yang diterima berdasarkan aturan yang ditetapkan dari controller. Controller menentukan apa yang harus dilakukan dengan paket dan jika perlu mengirimkan aturan baru untuk perangkat sehingga dapat menangani paket data di masa depan dengan cara yang sama (Indriani Lestariningati, 2018).

Pada saat ini, beberapa controller yang terkenal dan telah berkembang diantaranya NOX, POX, Floodlight, OpenDaylight, Open Network Operating

System (ONOS), dan RYU. Namun, sejumlah pengontrol dan varian rasa lain tersedia dalam literatur yang ada (Zhu et al., 2019). Pada penelitian ini menggunakan Controller RYU dan OpenDaylight.



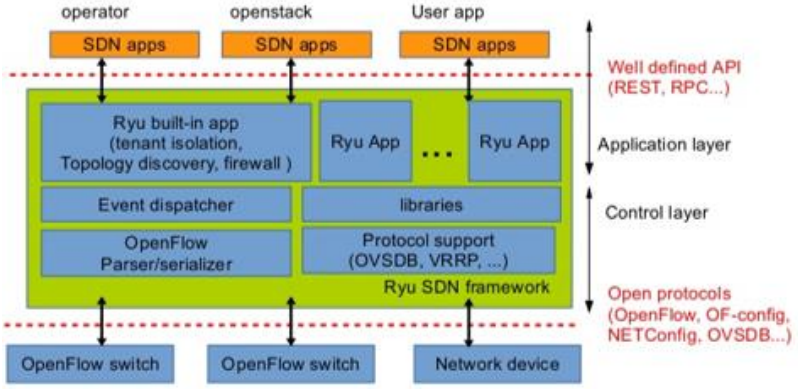
Gambar 2. 6 Arsitektur SDN Controller

2.4.1 RYU Controller

RYU adalah kerangka kerja jaringan SDN berbasis komponen sumber terbuka yang dikembangkan oleh *Nippon Telegraph dan Telephone (NTT)*. RYU adalah pengontrol SDN yang diimplementasikan sepenuhnya dengan phyton. Nama RYU berasal dari kata dalam bahasa Japan yang berarti “mengalir” yang dimana merupakan naga Japan, salah satu dewa air. Ia mengelola kontrol “aliran” untuk mengaktifkan kecerdasan jaringan. Controller RYU menyediakan komponen perangkat lunak dengan antarmuka program aplikasi (API) yang terdefinisi dengan baik sehingga dapat memudahkan pengembang untuk membuat aplikasi manajemen dan kontrol jaringan baru.

Controller RYU pada SDN memiliki tiga lapisan. Lapisan atas dikenal sebagai lapisan atas yang terdiri dari aplikasi logika bisnis dan jaringan. Lapisan tengah dikenal sebagai lapisan kontrol atau kerangka SDN yang terdiri dari layanan jaringan, sedangkan untuk lapisan bawah dikenal sebagai lapisan infrastruktur yang terdiri dari perangkat fisik dan virtual (Time & Time, 2020).

RYU mendukung berbagai protokol untuk mengelola perangkat jaringan, seperti OpenFlow, Netconf, OF-config dan lain sebagainya. Pada OpenFlow sendiri, RYU mendukung sepenuhnya 1.0, 1.2, 1.3, 1.4, 1.5 dan Nicira Extensions. Semua kode tersedia secara bebas di bawah lisensi Apache 2.0 Ryu berbasis bahasa phyton dan bersifat Open Source (Edgar et al., 2019).



Gambar 2. 7 Arsitektur Controller RYU

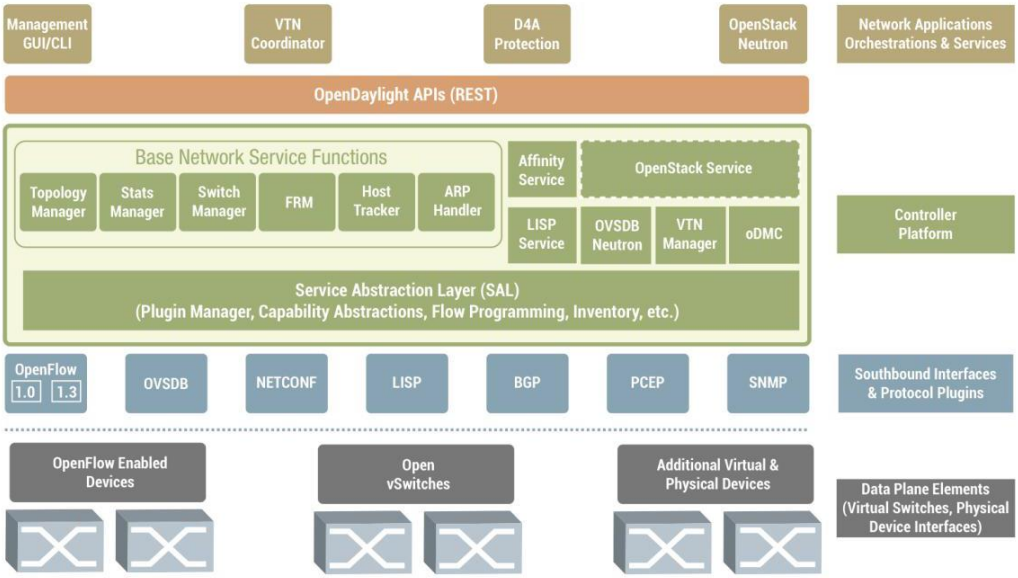
2.4.2 OpenDaylight

Dalam dunia jaringan yang terus berkembang, pengelolaan dan pengaturan jaringan yang efektif menjadi semakin penting. OpenDaylight menyediakan kerangka kerja yang fleksibel dan kuat untuk membangun solusi SDN yang *scalable* dan *interoperable*.

Dengan fitur-fiturnya yang canggih dan komunitas pengembang yang aktif, OpenDaylight tidak hanya memungkinkan pengelolaan jaringan yang lebih efisien tetapi juga mempercepat inovasi dalam teknologi jaringan. Dalam pengantar ini, kita akan mengeksplorasi konsep dasar OpenDaylight, fungsionalitasnya, dan peran pentingnya dalam merevolusi manajemen jaringan modern.

OpenDaylight merupakan Controller SDN yang dikembangkan oleh komunitas pengembang terbuka LINUX dan merupakan jawaban atas kebutuhan industri pada jaringan yang memiliki kemampuan pemrograman mandiri (Edgar et al., 2019). OpenDaylight didukung oleh IBM, Cisco, Juniper, VMWare dan beberapa vendor jaringan besar lainnya. OpenDaylight sendiri bersifat OpenSource dan diimpelemntasikan di Java.

OpenDaylight SDN Controller memiliki tiga lapisan. Lapisan atas terdiri dari aplikasi logis bisnis dan jaringan. Lapisan tengah merupakan lapisan kerangka kerja, sedangkan lapisan bawah terdiri dari perangkat fisik dan virtual. OpenDaylight bertujuan untuk mengurangi vendor, mengunci sehingga mendukung protokol selain OpenFlow dan BGP-LS sebagai plugin terpisah (Khattak et al., 2014)



Gambar 2. 8 Arsitektur Pengontrol OpenDaylight SDN

2.5 Miniet

Mininet merupakan sebuah emulator jaringan sumber terbuka yang memungkinkan pengguna untuk membuat dan menjalankan topologi jaringan virtual di dalam satu sistem komputer. Dengan kata lain, mininet memungkinkan anda membuat jaringan virtual yang terdiri dari berbagai perangkat seperti switch, router, dan host yang dapat diatur dan di uji tanpa memerlukan perangkat keras fisik.

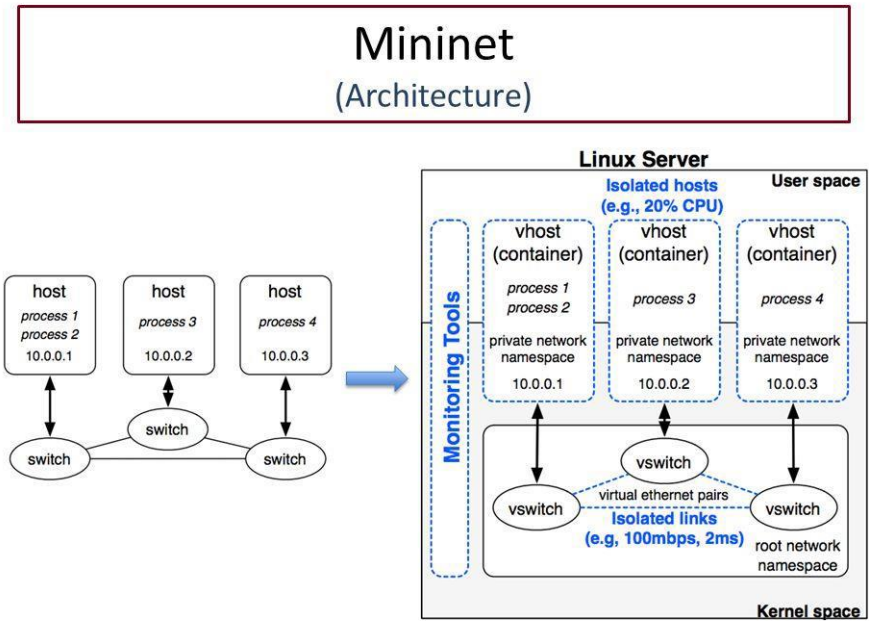
Mininet berfungsi sebagai emulator atau simulator jaringan, memungkinkan pengguna untuk membuat dan menguji topologi jaringan virtual tanpa harus memiliki perangkat keras jaringan fisik. Secara sederhana mininet ini berfungsi untuk emulasi pada bagian data path untuk mengetes konfigurasi jaringan SDN.

Sedangkan untuk melakukan testing pada mininet dapat dilakukan command “sudo mn” (Sudiyatmoko et al., 2016).

Mininet bersifat sumber terbuka, yang berarti kode sumbernya dapat dikases oleh siapa saja. Hal ini memungkinkan pengguna untuk memodifikasi dan mengadaptasi sesuai dengan kebutuhan mereka. Mininet berguna terutama untuk keperluan pengembangan dan pengujian di lingkungan pengembangan atau penelitian. Ini memberikan fleksibilitas kepada pengguna untuk menciptakan dan menguji berbagai skenario jaringan tanpa harus menghadapi kompleksitas dan biaya yang terkait dengan perangkat keras fisik.

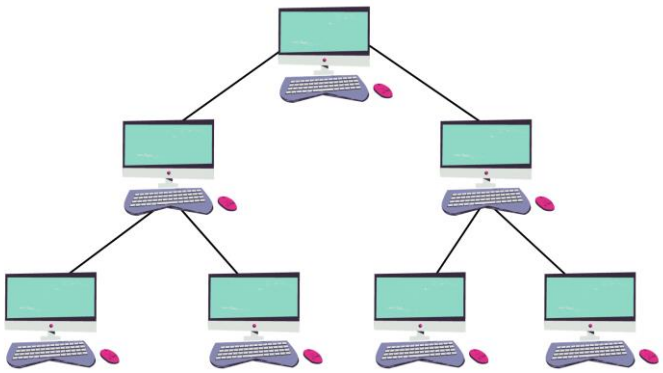
Mininet merupakan solusi yang dianggap paling unggul dalam hal kemudahan penggunaan, performansi, akurasi, dan skalabilitas. Ia mampu menyediakan lingkungan yang relastis dan nyaman dengan harga yang murah.

Mininet menggunakan pendeketan *light-wight* menggunakan virtualisasi level OS mencakup proses – proses dan namespace jaringan, sehingga memungkinkan dilakukannya simulasi jaringan dengan skala sangat besar (hingga ratusan node). Mininet dapat menciptakan jaringan virtual yang relastis, menjalankan real kernel, switch dan kode aplikasi, pada single machine baik berupak physical maupun virtual atau cloud (Kementrian Kesehatan Republik Indonesia, 2018).



Gambar 2. 9 Arsitektur Mininet

2.6 Topologi Jaringan Tree



Gambar 2. 10 Topologi Jaringan Tree

Topologi jaringan pohon atau tree network adalah suatu jenis konfigurasi jaringan komputer yang menggabungkan dua atau lebih struktur topologi berbeda. Dalam topologi ini, beberapa pohon atau cabang jaringan dihubungkan melalui satu titik pusat atau simpul utama yang disebut dengan root atau akar. Pusat ini berperan sebagai pengatur utama atau pemberi layanan kepada seluruh bagian jaringan.

Karakteristik utama dari topologi jaringan pohon melibatkan struktur hierarki, dimana setiap cabang atau pohon dari simpul utama dapat memiliki cabang – cabang lebih kecil lagi. Pusat atau simpul utama memainkan peran penting dalam pengiriman data antar bagian – bagian jaringan.

Keuntungan dari topologi pohon melibatkan kemudahan dalam manajemen dan organisasi, karena hierarki yang jelas membuatnya lebih muda untuk memahami dan mengelola bagian – bagian jaringan. Namun, kelemahannya termasuk kerentanan terhadap kegagalan dalam simpul utama. Jika simpul utama gagal, maka seluruh bagian jaringan yang terhubung dengannya dapat terpengaruh.

Topologi tree sering digunakan dalam jaringan bisnis dan organisasi besar karena kemampuannya untuk menyediakan struktur yang terorganisir dan mudah dikelola. Oleh karena itu, dalam memilih topologi jaringan, kita juga harus mempertimbangkan kebutusan spesifik dari suatu lingkungan dan potensi resiko kegagalan sistem.

2.7 Wireshark

Gambar 2. 13 Logo Wireshark

Wireshark adalah sebuah perangkat lunak open source yang berfungsi sebagai analisis paket jaringan. Ini berarti bahwa Wireshark dapat digunakan untuk memantau dan menganalisis data yang dikirim melalui jaringan komputer, termasuk data yang dikirim melalui kabel jaringan, jaringan nirkabel (Wi-Fi) dan berbagai protokol jaringan lainnya.

Dengan menggunakan Wireshark, pengguna dapat menangkap dan menganalisis paket data yang dikirim dan diterima oleh komputer atau perangkat jaringan lainnya. Hal ini memungkinkan pengguna pengguna untuk memahami interaksi antara berbagai perangkat dalam jaringan, mendeteksi masalah jaringan, memeriksa keamanan jaringan, serta memvalidasi kinerja jaringan.

Fitur utama dari Wireshark meliputi kemampuan untuk menangkap dan menyimpan paket data dalam berbagai format, menampilkan detail lengkap dari setiap paket termasuk informasi header dan payload, menyediakan filter untuk menampilkan paket yang spesifik, dan mendukung berbagai protokol jaringan seperti TCP, UDP, HTTP, DNS, dan banyak lagi.

Wireshark sering digunakan oleh administrator jaringan, insinyur jaringan, penelitian kemanan dan professional IT lainnya untuk memecahkan masalah jaringan, menganalisis kinerja, dan melakukan penelitian terkait dengan protokol jaringan. Keunggulan utama dari Wireshark adalah bahwa itu merupakan perangkat lunak sumber terbuka, sehingga dapat diunduh dan digunakan secara gratis oleh siapapun.

2.8 Internet Control Message Protokol (ICMP)

Internet Control Message Protocol (ICMP) merupakan salah satu protokol dalam suite protokol internet (TCP/IP) yang digunakan untuk mengirim pesan kontrol dan kesalahan antara perangkat dalam jaringan. ICMP bekerja diatas lapisan IP (Internet Protocol) dan digunakan untuk berbagai tujuan, termasuk :

1. Reporting Errors

   ICMP digunakan untuk melaporkan kesalahan yang terjadi selama pengiriman paket data, seperti ketika tujuan tidak dapat dicapai (host tidak dapat dijangkau), TTL (*Time-to-Live*) habis, atau ketika sebuah paket dibuang oleh router karena kelebihan beban.

2. Ping dan Traceroute

   Dua perintah yang paling umum digunakan yang berbasis ICMP adalah ping dan traceroute. Ping digunakan untuk menguji koneksi ke sebuah host dengan mengirimkan pesan ICMP Echo Request dan menerima balasan ICMP Echo Reply. Traceroute digunakan untuk melacak rute yang diambil oleh paket data melalui jaringan dengan mengirimkan serangan paket ICMP dan melacak respons dari setiap node dalam perjalanan

3. Konfigurasi da Pengujian Jaringan

   ICMP dapat digunakan untuk pengujian dan konfigurasi jaringan, seperti dengan pengujian kecepatan koneksi menggunakan perangkat lunak seperti *iperf* atau dengan pengujian ketersediaan host menggunakan alat monitoring jaringan

4. Pemberitahuan Mengenai Pemutusan Jaringan

   ICMP juga digunakan untuk memberikan pemberitahuan kepada pengguna atau perangkat lain dalam jaringan mengenai kondisi jaringan, seperti pemutusan jaringan yang tidak terduga atau keadaan darurat lainnya.

Meskipun ICMP memberikan banyak manfaat dalam manajemen dan pengoperasian jaringan, perlu diingat bahwa beberapa jenis serangan jaringan, seperti serangan ping atau serangan ICMP flood, juga dapat di eksploitasi menggunakan protokol ini. Oleh karena itu, dalam konfigurasi jaringan yang aman, ICMP mungkin perlu dikontrol dengan hati – hati.

2.9 Quality of Service (Qos)

Quality of Service (QoS) merupakan metode pengukuran tentang seberapa baik jaringan dan merupakan suatu usaha untuk mendefinisikan karakteristik dan sifat dari satu service. QoS digunakan untuk mengukur sekumpulan atribut kinerja yang telah dispesifikasikan dan diasosiasikan dengan suatu service (Cintasari, 2018).

Tujuan dari *QoS (Quality of Service)* adalah untuk memastikan kualitas pengalaman pengguna dalam jaringan komputer atau system komunikasi. QoS bertujuan untuk mengatur dan mengontrol lalu lintas data dalam jaringan dengan cara memprioritaskan aplikasi atau layanan yang lebih penting, serta memastikan pengiriman yang tepat waktu, keandalan dan kinerja yang memadai. Adapun parameter yang akan digunakan untuk analisis kinerja controller yaitu *throughput, latency dan packet loss.*

2.9.1 Throughput

Throughput yaitu jumlah data yang berhasil ditransfer dari satu titik ke titik lain dalam jaringan dalam waktu tertentu. Biasanya diukur dalam bit per detik (bps), kilobit per detik (kbps) atau megabit per detik (mbps).

Adapun persamaan perhitungan *throughput* :

$$Throughput = \frac{jumlah\ data\ yang\ diterima}{lama\ pengamatan}$$

Table 2. 1 Kategori Throughput

Kategori Degredasi	Throughput	Indeks
Sangat Baik	>100 Mbps	4
Baik	50-100 Mbps	3
Cukup	10-50 Mbps	2
Buruk	1-10 Mbps	1

Sumber: TIPHON

2.9.2 Latency

Latency adalah waktu yang dibutuhkan data untuk menempuh jarak dari asal ke tujuan. *Latency* dapat dipengaruhi oleh jarak, media fisik, kongesti atau juga waktu proses yang lama. Pada jaringan SDN, *latency* didefinisikan sebagai tanggapan yang dapat diberikan oleh controller pada setiap detiknya (Sulfiana, 2019).

Table 2. 2 Kategori Latency

Kategori Degredasi	Latency	Indeks
Sangat Baik	<50 ms	4
Baik	100-200 ms	3
Cukup	200-300 ms	2
Buruk	>300 ms	1

Sumber: TIPHON

2.9.3 Packet Loss

Packet loss merupakan suatu parameter yang menggambarkan suatu kondisi yang menunjukkan jumlah total paket yang hilang, dapat terjadi karena collison dan congestion pada jaringan. Hal ini berpengaruh pada semua aplikasi karena retransmisi akan mengurangi efesiensi jaringan secara keseluruhan meskipun jumlah bandwidth cukup tersedia untuk aplikasi – aplikasi tersebut.

Dalam SDN, *packet loss* merupakan jumlah paket yang hilang pada suatu jaringan yang disebabkan ketika satu atau lebih paket yang dikirim melalui jaringan IP gagal sampai ditempat tujuan (Sulfiana, 2019). Untuk mencari nilai dari *packet loss* dapat dihitung dengan persamaan :

$$Packet\ loss = \frac{(paket\ data\ dikirim - paket\ data\ diterima)}{paket\ data\ yang\ dikirim}\ x\ 100\%$$

Indeks dan kategori Packet loss dapat dilihat pada table 2.

Table 2. 3 Kategori Packet loss

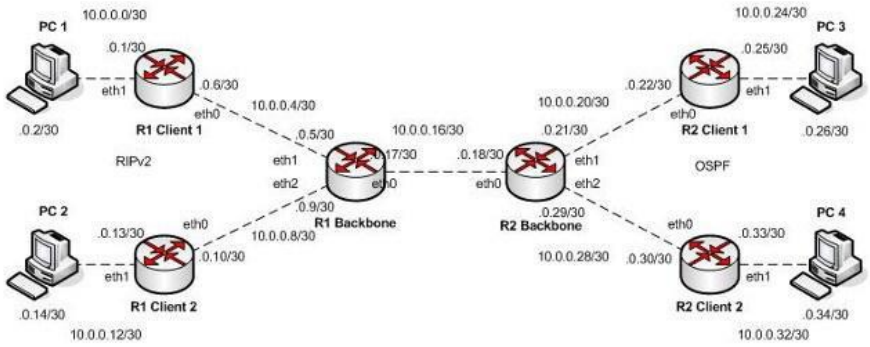
Kategori Degredasi	Packet Loss	Indeks
Sangat Baik	0 – 2%	4
Baik	3 – 14%	3
Cukup	15 – 24%	2
Buruk	>25%	1

Sumber: TIPHON

2.10 Iperf

Iperf adalah alat pengukuran bandwidth dan kinerja jaringan yang open-source dan lintas platform. Alat ini membantu dalam mengevaluasi dan menganalisis throughput jaringan TCP dan UDP. *Iperf* dirancang untuk memberikan hasil yang detail dan akurat, memungkinkan pengguna untuk memahami kemampuan jaringan mereka dengan lebih baik. Tool ini juga bersifat freeware. Tool ini hanya dapat dijalankan melalui command prompt dan tidak memiliki tampilan GUI (Stmik-amik-riau et al., 2016)

2.11 Routing Protokol OSPF (Open Shortest Path First)



Gambar 2. 14 Routing Protokol OSPF (Open Shortest Path First)

OSPF (Open Shortest Path First) merupakan sebuah routing protokol berjenis *IGP (Interior Gateway routing Potokol)* yang hanya dapat bekerja dalam jaringan internal suatu organisasi atau perusahaan. Jaringan internal maksudnya adalah jaringan dimana kita masih memeliki hak untuk menggunakan, mengatur dan memodifikasinya. Singkatnya, kita masih memiliki hak administrasi terhadap

jaringan tersebut. Selain itu, OSPF merupakan routing protokol berstandar terbuka (Ainy, 2017). Ada beberapa hal penting mengenai OSPF, diantaranya :

1. Link State Protocol

   OPSF adalah protokol routing berbasis state link, yang berarti setiap router dalam area OSPF mengumpulkan informasi tentang jaringan dan topologi yang disebarkan oleh router lain. Informasi ini digunakan untuk membangun database topo logi jaringan.
2. Perhitungan Rute Terpendek

   OSPF menggunakan algoritma djikstra untuk menghitung rute terpendek (shortest path) antara router dalam jaringan. Ini memungkinkan OSPF untuk menemukan rute yang paling efesien antara dua titik dalam jaringan.
3. Hierarchical Design

   OSPF dibagi menjadi area – area yang membentuk hierarki. Ini memungkinkan OSPF untuk skalabel dalam jaringan yang besar, dengan mengurangi overhead dalam pertukaran dalam informasi topologi.
4. Konvergensi Cepat

   OSPF dirancang untuk memiliki waktu konvergensi yang cepat, yang berarti bahwa jika terjadi perubahan dalam topologi jaringan, OSPF akan secara cepat menyesuaikan rute – rute jaringan untuk menghindari paket yang hilang atau terlewat.
5. Authentication dan Security

   OSPF mendukung berbagai mekanisme otentikasi untuk memastikan keamanan dalam pertukaran informasi routing antara router, termasuk plaintext, MD5, dan metode otentikasi kunci public.

OSPF banyak digunakan dalam jaringan besar, terutama dalam jaringan korporat, ISP (*Internet Service Provider*) dan organisasi yang memiliki kebutuhan routing yang kompleks. Dengan keandalan, kecepatan konvergensi dan kemampuan skalabilitasnya, OSPF menjadi salah satu protokol routing yang paling umum digunakan dalam pengaturan jaringan yang luas.

2.12 Quagga

Quagga adalah sebuah suite software routing yang digunakan untuk mengimplementasikan protokol routing pada sistem operasi berbasis Unix, seperti Linux dan BSD. Quagga merupakan fork dari proyek Zebra yang sudah tidak aktif, dan menawarkan fungsionalitas routing yang canggih untuk jaringan skala besar maupun kecil. Adapun fitur dari quagga :

1. **Implementasi Protokol Routing:** Quagga mendukung beberapa protokol routing standar industri, termasuk :
   - OSPF (Open Shortest Path First) versi 2 dan 3
   - RIP (Routing Information Protocol) versi 1 dan 2
   - BGP (Border Gateway Protocol) versi 4
   - IS-IS (Intermediate System to Intermediate System)
2. **Arsitektur Modular:** Quagga memiliki arsitektur modular di mana setiap protokol routing diimplementasikan sebagai daemon yang terpisah. Ini memungkinkan fleksibilitas dalam mengonfigurasi dan mengelola protokol routing sesuai kebutuhan jaringan.
3. **CLI (Command Line Interface):** Quagga menyediakan antarmuka baris perintah (CLI) yang mirip dengan perintah-perintah Cisco IOS, memudahkan administrasi dan konfigurasi bagi mereka yang sudah familiar dengan perangkat Cisco.
4. **Integrasi dengan Kernel:** Quagga berfungsi sebagai pengguna ruang (user space) daemon yang mengelola tabel routing kernel, memungkinkan pembaruan dinamis tabel routing berdasarkan informasi yang diterima dari protokol routing.

Quagga memiliki berbagai kegunaan dalam manajemen jaringan dan implementasi protokol routing, menjadikannya alat yang sangat berguna dalam berbagai skenario jaringan. Bebetapa kegunaan utama dari Quagga adalah sebagai berikut :

1. Routing Dinamis di Jaringan Enterprise

   Quagga memungkinkan administrator jaringan untuk mengimplementasikan routing dinamis menggunakan berbagai protokol routing seperti OSPF, BGP, dan RIP. Ini sangat berguna dalam jaringan

perusahaan yang kompleks di mana rute optimal mungkin perlu diperbarui secara dinamis untuk mengakomodasi perubahan jaringan, kegagalan perangkat, atau optimasi jalur.

2. Interkoneksi Jaringan yang Berbeda

   Dalam lingkungan yang menggabungkan beberapa jaringan atau subnet yang berbeda, Quagga dapat digunakan untuk menghubungkan dan mengelola rute antar jaringan tersebut. Protokol seperti OSPF dan BGP membantu memastikan bahwa data dapat mengalir dengan lancar antara jaringan yang berbeda, bahkan jika mereka menggunakan skema alamat yang berbeda.

3. Penggunaan dalam Penelitian dan Pengembangan

   Quagga sering digunakan dalam penelitian jaringan dan pengembangan teknologi baru. Karena sifatnya yang open-source dan fleksibel, Quagga memungkinkan peneliti untuk menguji dan mengembangkan algoritma routing baru, mempelajari perilaku protokol routing, atau mensimulasikan skenario jaringan yang kompleks.

4. Routing di Jaringan ISP

   ISP (Internet Service Providers) menggunakan Quagga untuk mengelola routing BGP, yang merupakan protokol routing utama yang digunakan untuk pertukaran rute antar AS (*Autonomous Systems*) di internet. Quagga membantu ISP mengelola tabel routing besar, mengoptimalkan jalur, dan menjaga keandalan serta efisiensi jaringan mereka.

5. Manajemen Routing di Datacenter

   Data center yang besar dan kompleks sering kali memerlukan routing yang canggih untuk memastikan ketersediaan dan efisiensi layanan. Quagga dapat digunakan untuk mengelola routing internal dalam data center, memastikan bahwa lalu lintas data dapat mengalir secara optimal antara server, storage, dan perangkat jaringan lainnya.

6. Penggantian Perangkat Keras Routing Mahal

   Dengan menggunakan Quagga, perusahaan dan organisasi dapat menghemat biaya dengan mengurangi ketergantungan pada perangkat keras routing yang mahal. Quagga memungkinkan penggunaan server standar

untuk menjalankan fungsi routing, yang dapat mengurangi biaya operasional dan meningkatkan fleksibilitas jaringan.

7. Manajemen dan Monitoring Jaringan

   Quagga menyediakan alat dan antarmuka untuk memonitor dan mengelola tabel routing dan status protokol routing. Ini termasuk antarmuka CLI (*Command Line Interface*) yang mirip dengan Cisco IOS, yang memudahkan administrator jaringan untuk melakukan konfigurasi dan pemantauan.

8. Virtualisasi dan Jaringan Definisi Perangkat Lunak (SDN)

   Dalam lingkungan virtualisasi dan SDN, Quagga dapat digunakan untuk mengimplementasikan routing yang fleksibel dan dinamis. Dengan integrasi dengan platform virtualisasi dan alat SDN, Quagga membantu dalam membangun jaringan yang dapat dengan cepat beradaptasi dengan perubahan kebutuhan aplikasi dan beban kerja.

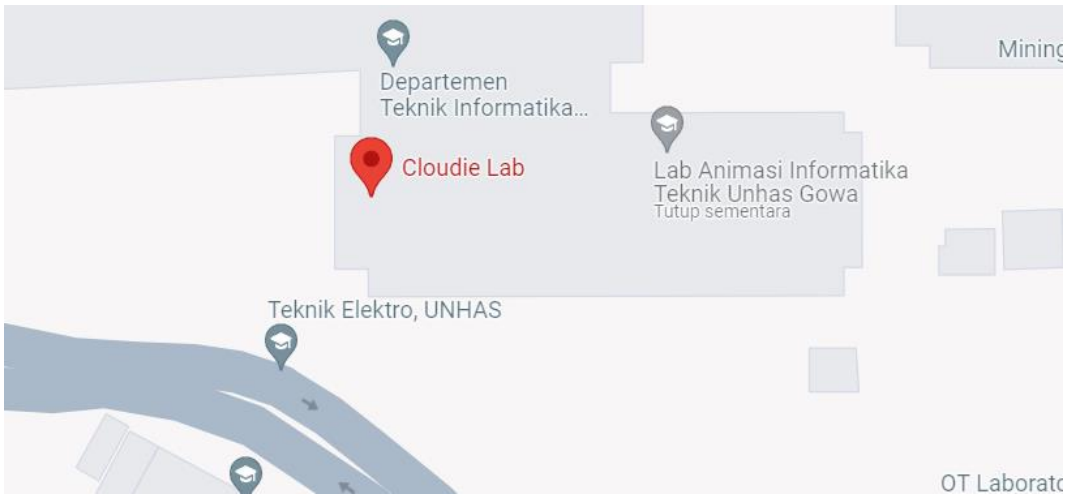
2.13 Zebra

Zebra adalah perangkat lunak routing open-source yang penting dalam sejarah pengembangan perangkat lunak routing dinamis. Meskipun pengembangannya telah dilanjutkan oleh Quagga dan FRRouting (FRR), konsep dan arsitektur dasar Zebra tetap menjadi fondasi dari banyak solusi routing modern. Dengan dukungan untuk berbagai protokol routing dan arsitektur yang fleksibel, Zebra (dan penerusnya) terus menjadi alat yang berguna untuk manajemen jaringan dinamis.

BAB 3
METODE PENELITIAN PERANCANGAN

3.1 Lokasi Penelitian

Proses penelitian dilakukan mulai Desember 2023 bertempat di Laboratorium Cloud Computing and Internet Engineering, Departemen Teknik Informatika, Fakultas Teknik, Universitas Hasanuddin.



Gambar 3. 1 Lokasi Penelitian

3.2 Instrumen Penelitian

Perangkat keras dan perangkat lunak yang digunakan sebagai pendukung penelitian ini antara lain :

3.2.1 Perangkat Keras

- ASUS TUF Gaming FX505DT_FX505DT (AMD Ryzen 7 3750 H with Radeon Vega Mobile Gfx (8 CPUs), ~2.3GHz), 8192 MB RAM

3.2.2 Perangkat Lunak

- Ubuntu 22.04
- Visual Studio Code 1.90.1
- Mininet 2.3.1b4
- RYU Controller 4.34

- Opendaylight Oxygen 0.8.4
- Protokol Routing OSPF
- Quagga 1.2.4
- Wireshark 3.2.3
- Xterm 3.5.3

3.3 Prosedur Penelitian

Tahapan pada penelitian ini sebagaimana ditunjukkan pada gambar dibawah ini :

Gambar 3. 2 Tahap Penelitian

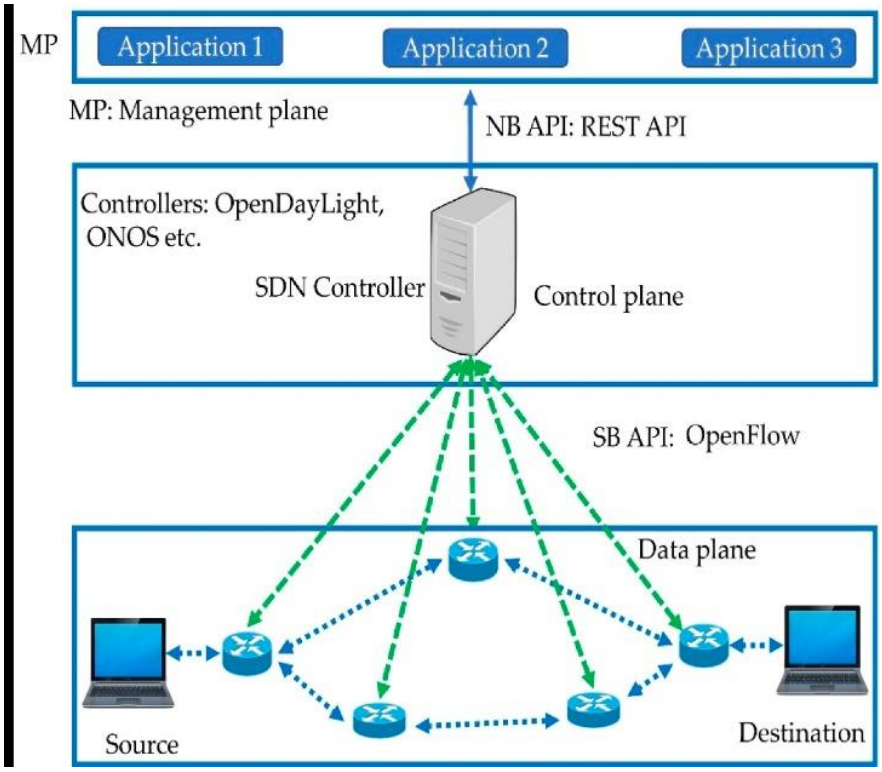
Tahap yang ada pada gambar 3.2 dijelaskan sebagai berikut :

1. Studi literatur dilakukan dengan mencari beberapa jurnal, e-book, artikel ilmiah serta sumber dari situs internet yang berkaitan dengan *Software Defined Network (SDN)*, controller, parameter pengujian performansi jaringan, serta konfigurasi mininet, protokol OSPF, OpenDaylight dan RYU.
2. Diskusi dan konsultasi dilakukan dengan Dosen Pembimbing mengenai penelitian yang akan dilakukan.
3. Penentuan kebutuhan sistem mencakup dari penentuan spesifikasi dari perangkat keras dan perangkat lunak yang akan digunakan.
4. Setelah penentuan kebutuhan sistem dilakukan, maka dilakukan instalasi tools dan emulator yang akan digunakan pada penelitian ini.
5. Setelah tools dan emulator terinstal, maka dilakukan konfigurasi antara tools dan emulator yang akan digunakan.
6. Setelah dilakukan tahap instalasi dan konfigurasi tools serta aplikasi emulasi, selanjutnya merancang topologi sesuai dengan skenario yang akan digunakan dalam pengujian.
7. Pengujian jaringan dilakukan untuk mengetahui implementasi OSPF pada jaringan SDN serta mengukur kinerja jaringan dengan parameter QoS.
8. Setelah pengujian jaringan, maka dilakukan analisis hasil dari pengujian jaringan yang telah dilakukan.

3.4 Gambaaran Umum Sistem

3.4.1 Integrasi OSPF dengan SDN

Salah satu inovasi yang paling signifikan adalah *Software Defined Network* (SDN), yang menawarkan arsitektur jaringan yang terpusat dan dapat diprogram. Seiring dengan itu, protokol routing seperti *Open Shortest Path First* (OSPF) tetap menjadi tulang punggung dalam mengelola rute data pada jaringan yang luas dan kompleks. Mengintegrasikan OSPF dengan SDN merupakan langkah strategis untuk memanfaatkan keunggulan kedua teknologi ini secara optimal.



Gambar 3. 3 Struktur OSPF pada SDN

Dimana OSPF nantinya terletaknya pada control plane. OSPF bekerja untuk pertukaran informasi topologi, perhitungan jalur terpendek, dan pengambilan keputusan routing/jalur pengiriman data jaringannya. OSPF lah yang mengatur lalu lintas di data plane berdasarkan perintah dari controllernya. southbound interface yang digunakan berupa open flow. Tugas dari southbound interface sebagai jembatan menghubungkan SDN Controller pada control plane dengan perangkat keras jaringan yang ada di data plane.

Komponen Utama :

1. **SDN Controller** : Dalam konteks ini, controller seperti RYU atau OpenDaylight akan menjalankan OSPF. Controller ini memiliki pandangan menyeluruh dari topologi jaringan dan bisa mengimplementasikan kebijakan routing secara terpusat.
2. **Switch SDN** : Switch dalam jaringan SDN hanya berfungsi untuk meneruskan paket berdasarkan aturan yang diberikan oleh controller. Switch ini tidak menjalankan OSPF secara mandiri tetapi menerima tabel routing dari controller.

3. **Northbound API** : API ini memungkinkan aplikasi di atas controller untuk berinteraksi dengan dan mengontrol jaringan. Ini dapat digunakan untuk mengkonfigurasi kebijakan routing dan mengumpulkan informasi jaringan.
4. **Southbound API**: API ini digunakan oleh controller untuk berkomunikasi dengan perangkat jaringan (switch). OpenFlow adalah salah satu protokol yang paling umum digunakan untuk komunikasi ini.

3.4.2 Prinsip Kerja OSPF

1. **Link-State Advertisements (LSAs)** : OSPF bekerja dengan cara mengumpulkan informasi status link dari router dalam jaringan dan mengirimkannya ke semua router lain dalam area OSPF. Informasi ini dikenal sebagai LSAs.
2. **Link-State Database (LSDB)**: Semua LSAs yang diterima oleh router disimpan dalam database yang dikenal sebagai LSDB. Semua router dalam satu area OSPF akan memiliki LSDB yang identik.
3. **Algoritma Dijkstra**: OSPF menggunakan algoritma Dijkstra untuk menghitung rute terpendek dari satu router ke semua tujuan lainnya dalam jaringan.

3.4.3 Konfigurasi Mininet dan Controller SDN

Pada tahap ini, mengkonfigurasi server mininer agar dapat terhubung dengan controller RYU dan OpenDaylight. Konfiguras dinyatakan berhasil apabila server mininet menjalankan suatu topologi jaringan, pada server controller akan terdeteksi perangkat jaringan pada switch yang telah dibuat oleh mininet. Apabila controller tidak dapat mendeteksi identitas perangkat jaringan maka proses akan diulang. Instalasi controller dan mininet sangat diperlukan pada tahap ini.

3.4.3.1 Instalasi Controller RYU

Adapun tahapan instalasi controller Ryu sebagai berikut :

1. Sebelum menginstal controller RYU, terlebih dahulu kita harus menginstal Phyton dan pip (package installer untuk Phyton) di system. Kita dapat menginstal pip menggunakan perintah :

~$ sudo pip install .

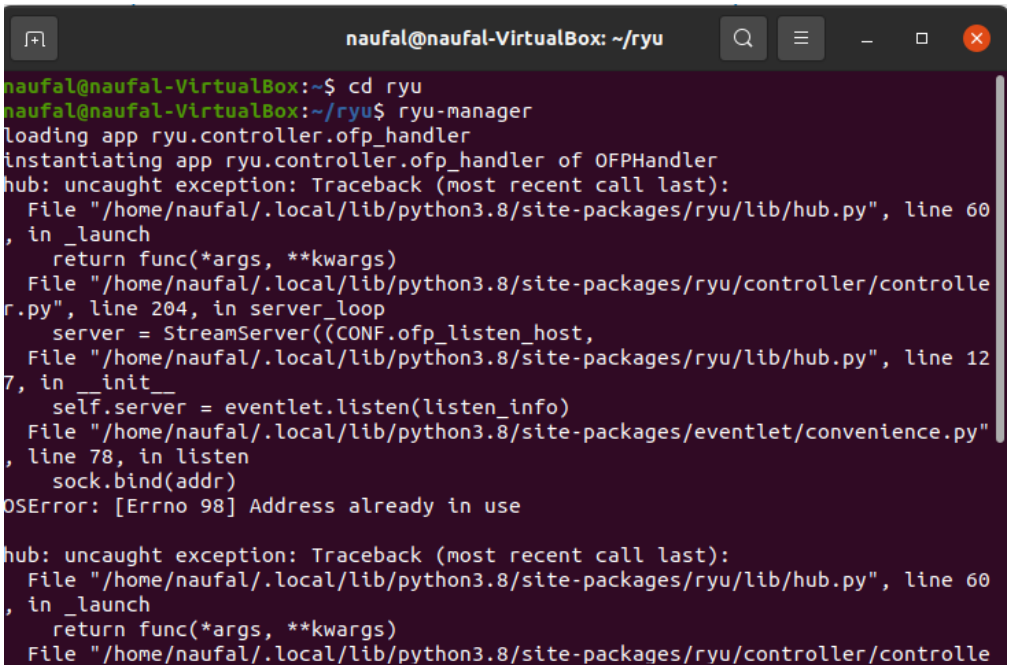
Setelah pip terinstal, kita dapat menggunakan perintah berikut untuk menginstal controller RYU :

~$ pip install ryu

2. Untuk menghindari masalah dependency, disarankan untuk menggunakan virtual environment. Kita dapat membuat lingkungan virtual dengan phyton -m venv myenv dan mengaktifkannya dengan source myenv/bin/activate. Kemudian, untuk meninstal RYU kita dapat menggunakan pip.
3. Setelah instalasi pip dan pyenv selesai, maka jalankan perintah :

   ~$ryu-manager

   Ini akan menampilkan menu bantuan RYU yang menunjukkan bahwa RYU erhasil di install seperti gambar dibawah ini :



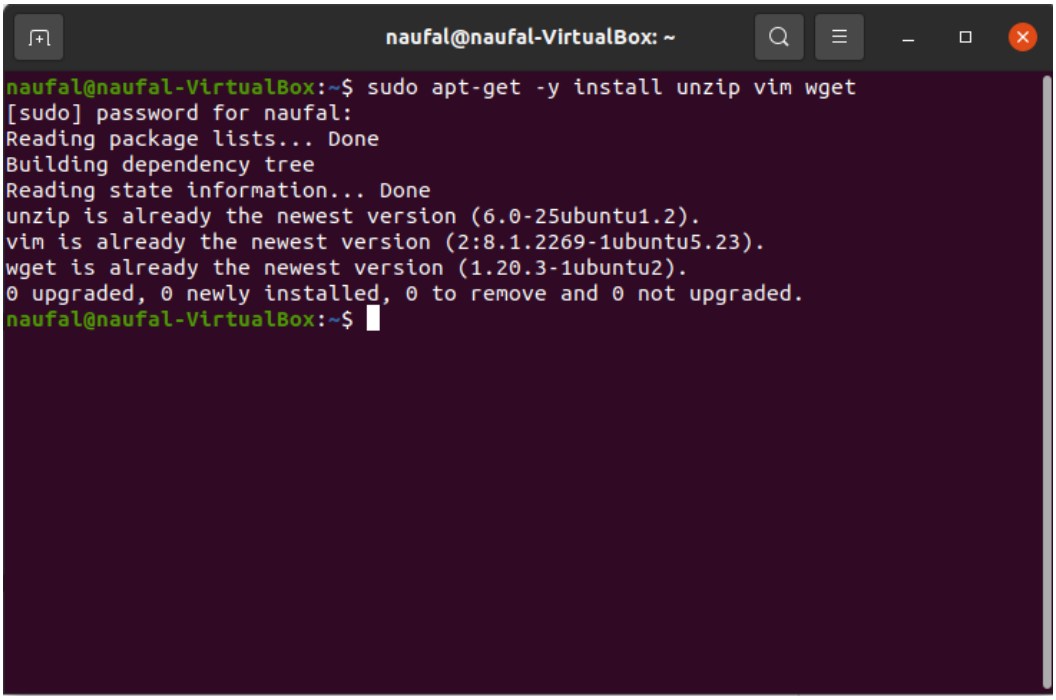
Gambar 3. 4 Tampilan RYU Saat Berhasil di Instal

3.4.3.2 Instalasi Controller OpenDaylight

Adapun tahapan instalasi controller OpenDaylight sebagai berikut :

1. Terlebih dahulu kita harus menginstal dependency, dengan menggunakan perintah berikut :

   ~$ sudo apt-get -y install unzip vim wget

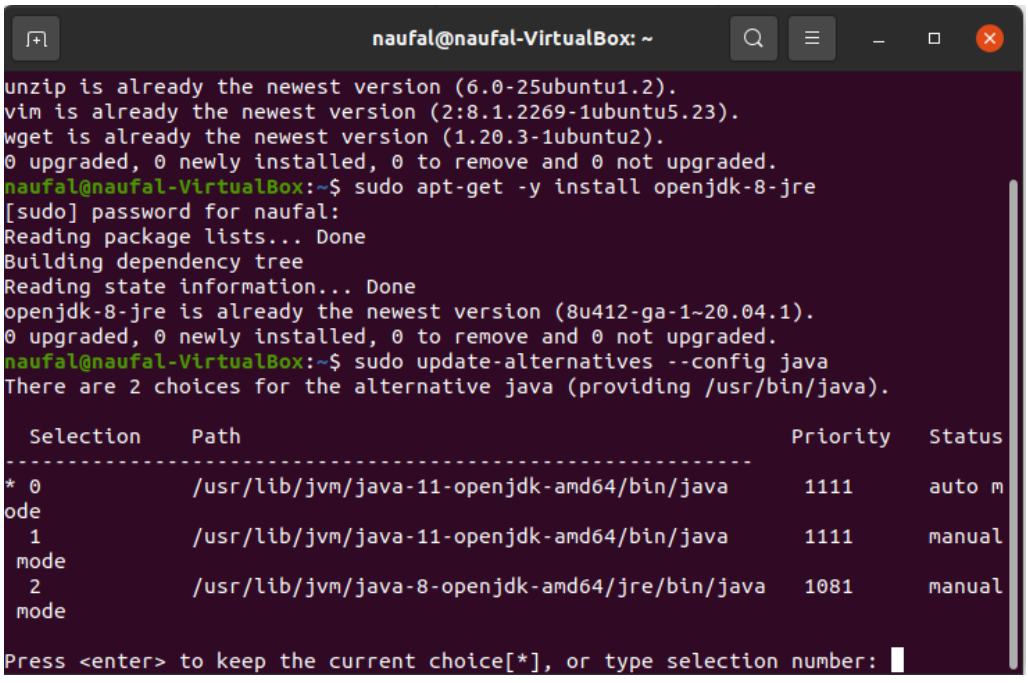


Gambar 3. 5 Instalasi Dependency

2. Karena OpenDaylight basenya dari java, maka kita gunakan JAVA *jre*. Untuk menginstal JAVA *jre*, kita dapat menggunakan perintah :

   ~$ sudo apt-get -y install openjdk-8-jre

   ~$ sudo update-alternatives --config java



Gambar 3. 6 Instalasi JAVA *jre*

3. Setelah menginstal JAVA jre, kemudian di export dan menyalakan bashrc. Dapat menggunakan perintah berikut :

   ~$ sudo echo 'export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre' >> ~/.bashrc

   ~$ source ~/.bashrc

4. Cek JAVA home sudah sesuai atau belum, dengan menggunakan perintah berikut :

   ~$ echo $JAVA_HOME

5. Selanjutnya mendownload controller OpenDaylight. Pada tahap ini, peneliti menggunakan OpenDaylight Oxygen versi 0.8.4. Dapat di download di link resmi OpenDaylight. Untuk mendownload OpenDaylight dapat menggunakan file tar atau zip. Dapat menggunakan perintah sebagai berikut :

   ~$wgethttps://nexus.opendaylight.org/content/repositories/opendaylight.release/org/opendaylight/integration/karaf/0.8.4/karaf-0.8.4.zip

6. Untuk membuat direktori karaf untuk lokasi instalasi OpenDaylight, dapat menggunakan perintah :

   ~$ sudo mkdir /usr/local/karaf

7. Untuk memindahkan OpenDaylight ke karaf dan unzip karaf, dapat menggunakan perintah berikut :

   ~$ sudo mv karaf-0.8.4.zip /usr/local/karaf

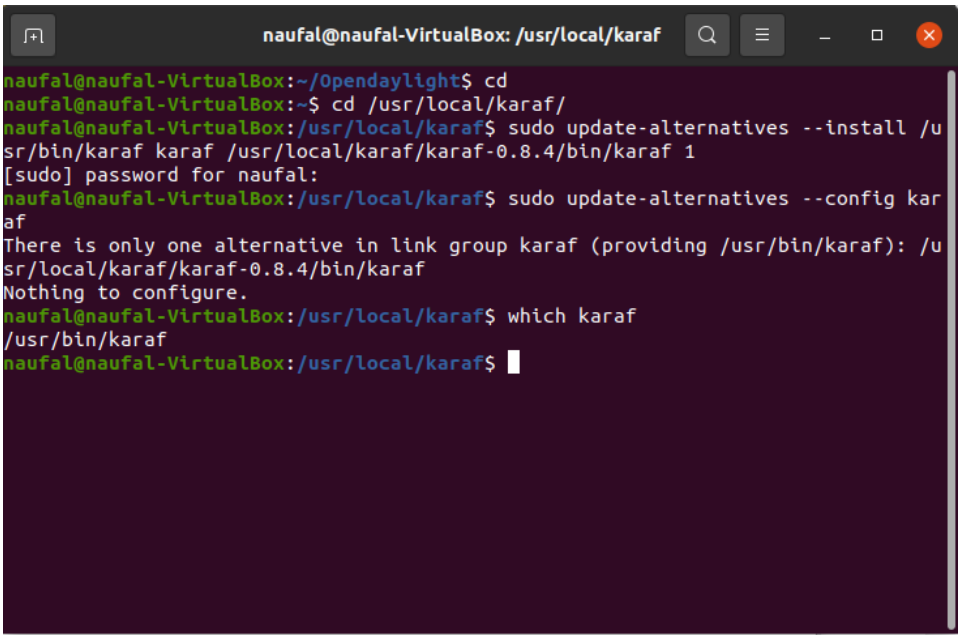
   ~$ sudo unzip /usr/local/karaf/karaf-0.8.4.zip -d /usr/local/karaf/

8. Selanjutnya instalasi karaf atau OpenDaylight, dapat menggunakan perintah :

   ~$ sudo update-alternatives --install /usr/bin/karaf karaf /usr/local/karaf/karaf-0.8.4/bin/karaf 1

   ~ $ sudo update-alternatives --config karaf

   ~$ which karaf



Gambar 3. 7 Instalasi karaf dan OpenDaylight

9. Setelah proses download dan instalasi OpenDaylight, selanjutnya menajalankan OpenDaylight dengan perintah :

   ~$ sudo -E karaf

Gambar 3. 8 Tampilan OpenDaylight

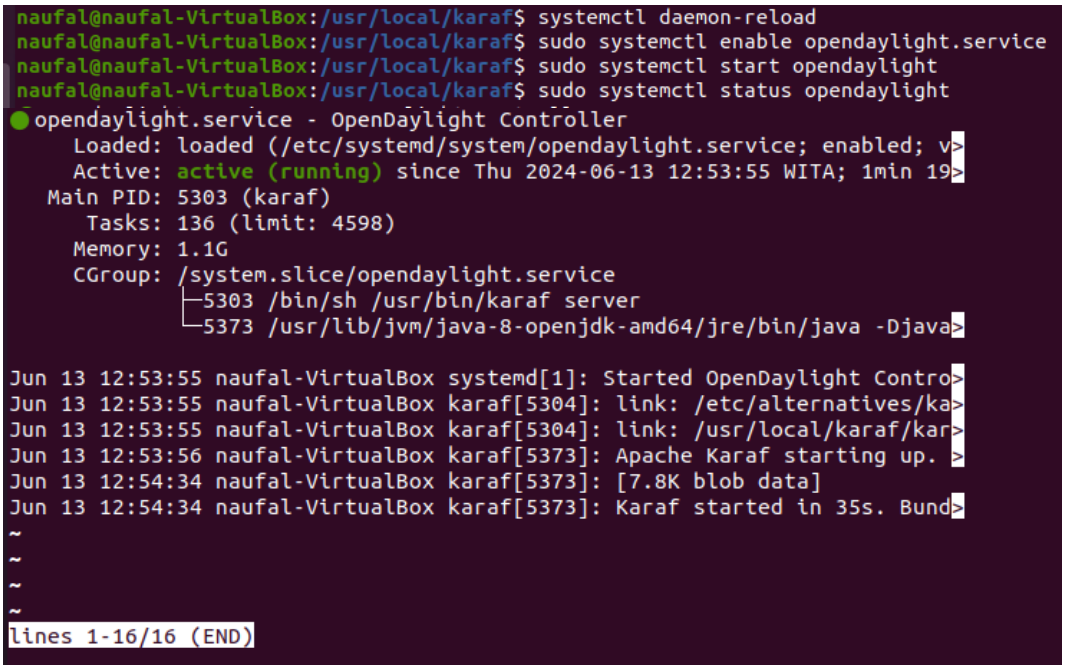
10. Untuk mengaktifkan dan reload OpenDaylight service, dapat menggunakan perintah :

~$ systemctl daemon-reload

~$ sudo systemctl enable opendaylight.service

~$ sudo systemctl start opendaylight

~$ sudo systemctl status opendaylight



Gambar 3. 9 Mengaktifkan dan Reload Opendaylight Service

3.4.3.3 Instalasi Mininet

Mininet merupakan alat simulasi jaringan yang memungkinkan pengguna untuk membuat, menjalankan dan menguji jaringan virtual yang terdiri dari host, switch, dan router. Fungsi dari mininet sebagai data plane atau yang bertanggung jawab untuk meneruskan paket data antar node. Adapun cara instalasi mininet seperti berikut :

1. Menginstal packages yang diperlukan dengan menggunakan perintah :

   ~$ sudo apt-get update

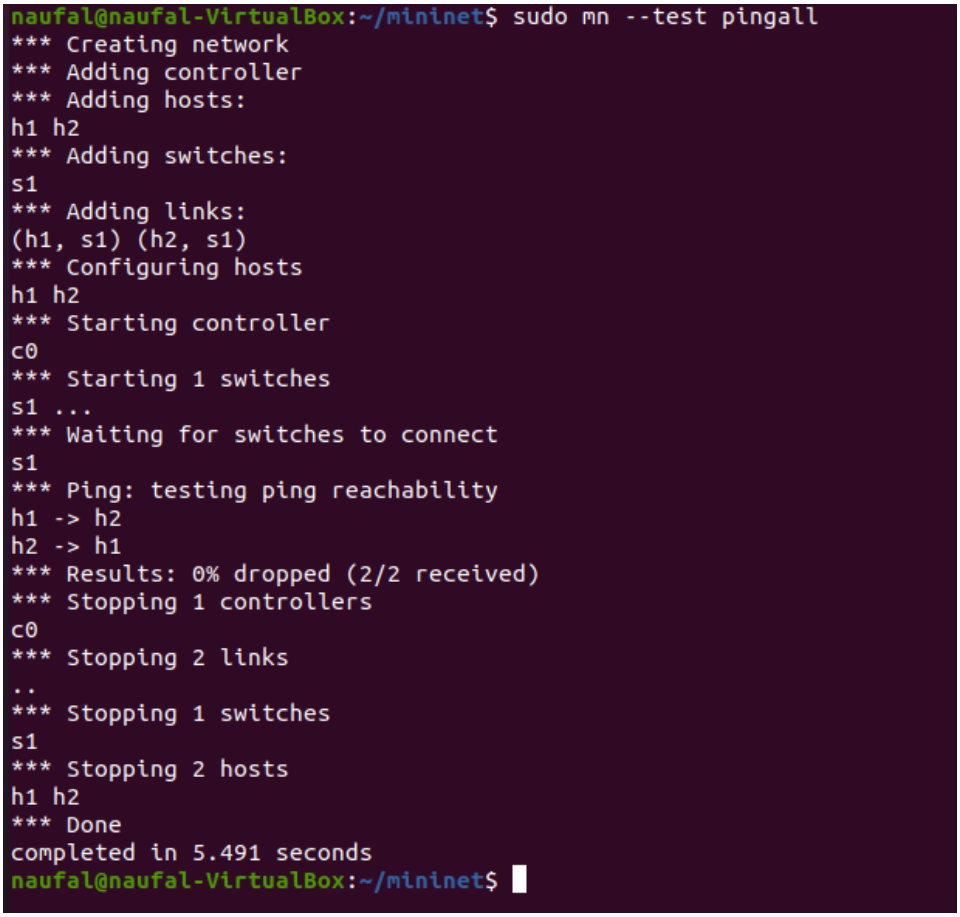
   ~$ sudo apt-get install git python-dev libssl-dev python-pip libffi-dev
2. Clone repository mininet dari github :

   ~$ git clone git://github.com/mininet/mininet
3. Masuk ke dalam direktori dan install dengan menggunakan perintah :

   ~$ cd mininet

   ~$ sudo make install
4. Tes hasil dari instalasi

   ~$ sudo mn --test pingall



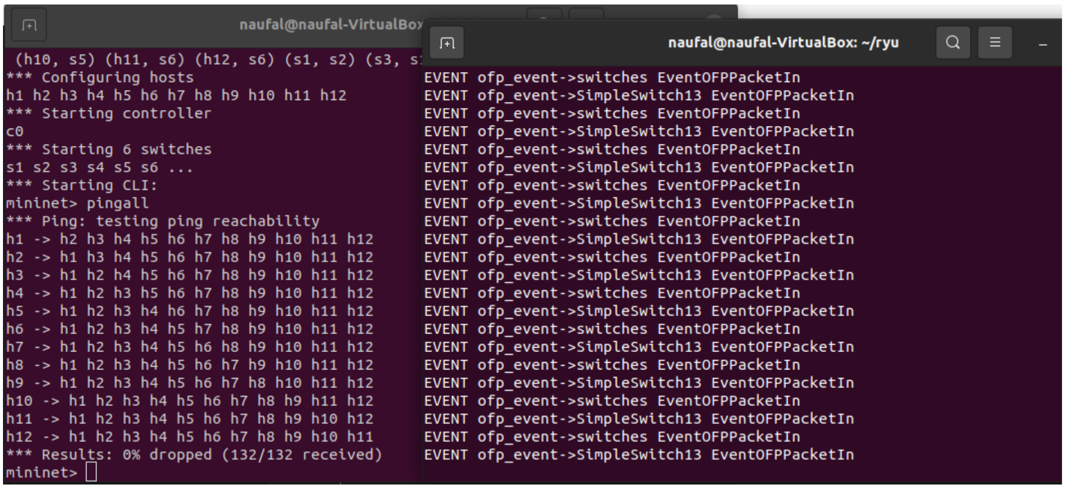
Gambar 3. 10 Hasil Instalasi Mininet

3.4.3.4 Menghubungkan Mininet dan RYU

Untuk menghubungkan mininet dan RYU, dapat dilakukan dengan cara :

1. Instal mininet dan RYU controller
2. Mulai mininet di terminal dengan topologi yang diinginkan (misalnya 6 switch dan 12 host)
3. Pada terminal lain, arahkan ke direktori RYU controller dan mulai controller dengan perintah :

   ~$ ryu-manager <controller file.py>
4. Pada mininet, gunakan perintah “xterm” untuk membuka terminal untuk setiap host dan switch
5. Pada terminal mininet, gunakan perintah “pingall” untuk menguji konektivitas jaringan.

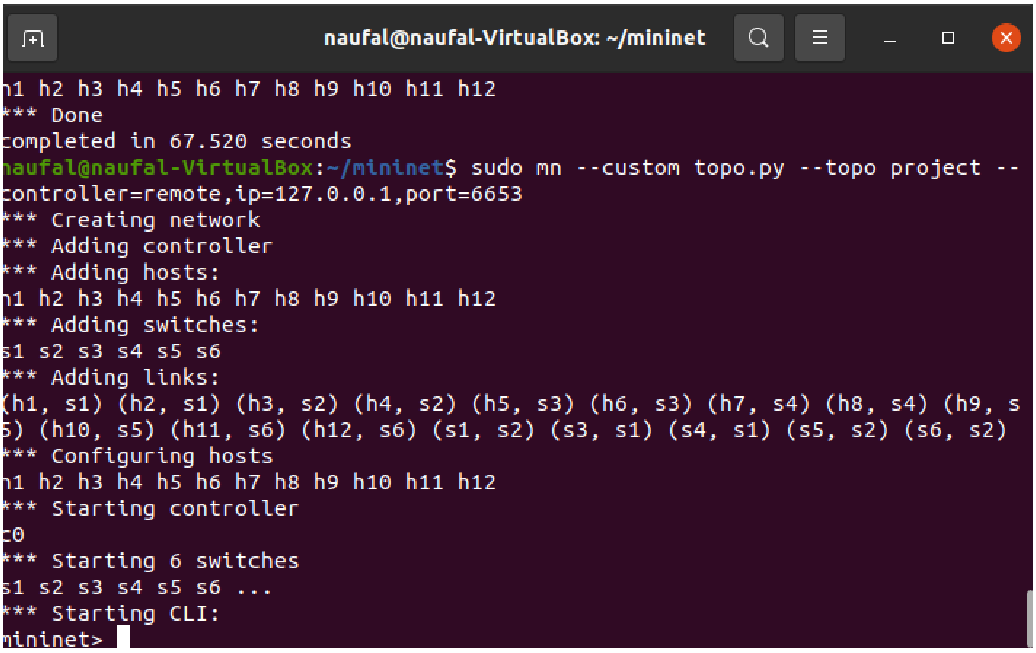


Gambar 3. 11 Integrasi Mininet dan RYU

3.4.3.5 Menghubungkan Mininet dan OpenDaylight

Untuk menghubungkan mininet dan OpenDaylight, dapat dilakukan dengan cara :

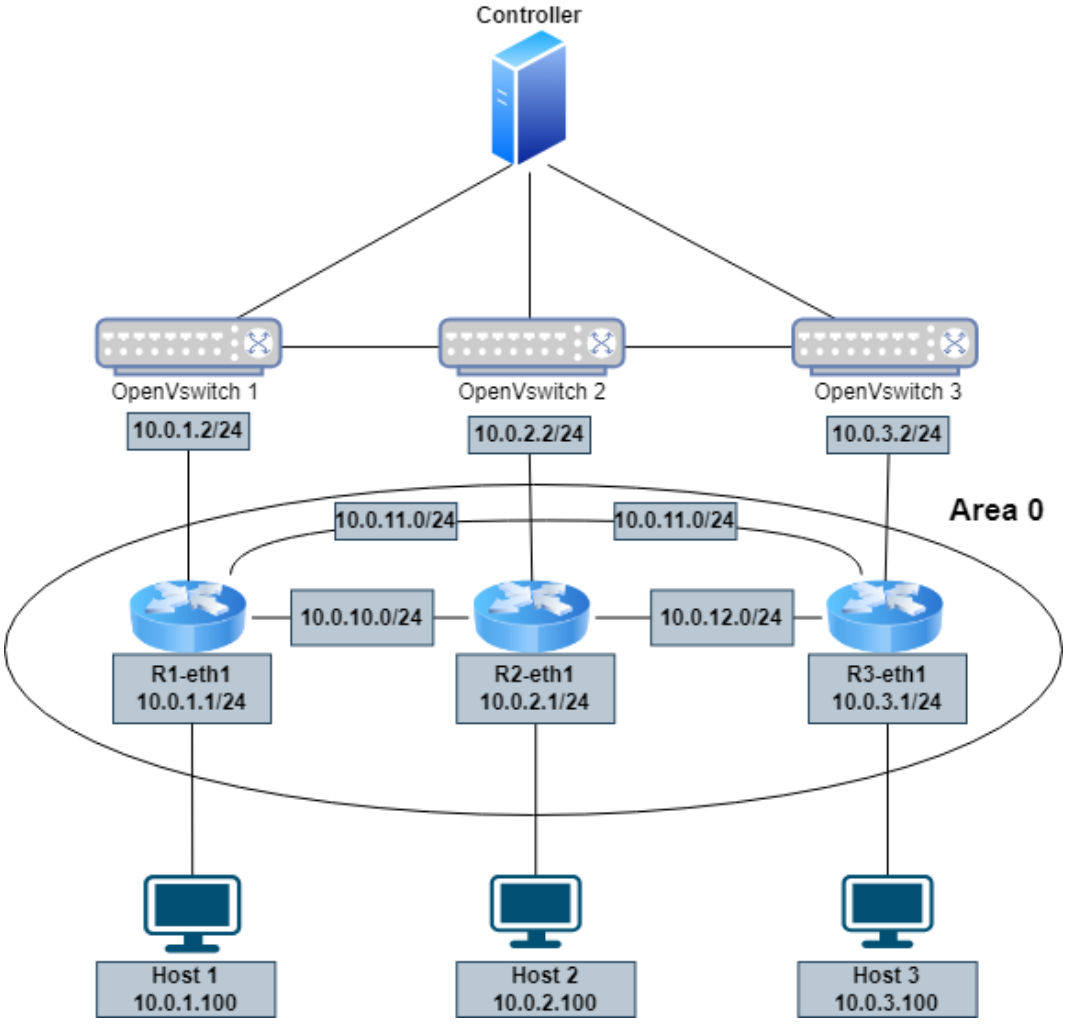
1. Instal mininet dan OpenDaylight
2. Jalankan controller OpenDaylight pada terminal
3. Mulai mininet di terminal dengan menggunakan perintah “sudo mn --custom <controller file.py> --topo <file yang ada di (controller file.py).py> kemudian arahkan controller dan port default ke arah Opendaylight dengan perintah --controller=remote,ip=127.0.0.1,port=6653



Gambar 3. 12 Intergrasi Mininet dan OpenDaylight

3.5 Perancangan Topologi

Setelah instalasi dan konfigurasi selesai dilakukan, selanjutnya merancang topologi yang akan digunakan dalam pengujian. Adapun skenario rancangan topologi sebagai berikut :



Gambar 3. 13 Topologi Jaringan

Adapun topologi yang digunakan adalah topologi tree dengan single area (area 0) yang terdiri dari 3 switch, 3 router serta 3 host yang saling terhubung. Topologi jaringan ini dijalankan pada mininet. Untuk mengukur parameter guna menentukan stabilitas jaringan dan tingkat Qos, kita perlu memiliki beberapa jalur untuk mencapai tujuan yang sama. Maka dibuatkanlah topologi seperti pada gambar 3.13

3.6 Konfigurasi Zebra

Setelah membuat rancangan topologi, konfigurasi zebra berdasarkan jumlah router yang dibuat pada topologi. Adapun cara untuk mengkonfigurasikan zebra pada router, dapat dilakukan dengan perintah :

~$ cd /usr/local/etc

~$ cp zebra.conf.sample r1zebra.conf

~$ cp zebra.conf.sample r2zebra.conf

~$ cp zebra.conf.sample r3zebra.conf

Untuk menjalankan zebra dapat melakukan dengan perintah :

~$ sudo zebra -d

3.7 Konfigurasi OSPF

Untuk menjalankan OSPF, dilakukan konfigurasi OSPF sesuai dengan jumlah router yang ada pada topologi. Konfigurasi dapat dilakukan dengan menggunakan perintah :

~$ cd /usr/local/etc

~$ cp ospfd.conf r1ospfd.conf

~$ cp ospfd.conf r2ospfd.conf

~$ cp ospfd.conf r3ospfd.conf

Edit r1ospfd.conf :

```
  GNU nano 4.8                                                r1ospfd.conf
hostname r1
password zebra
enable password zebra

interface r1-eth1
 ip ospf hello-interval 1
 ip ospf dead-interval 4

interface r1-eth2
 ip ospf hello-interval 1
 ip ospf dead-interval 4

interface r1-eth3
 ip ospf hello-interval 1
 ip ospf dead-interval 4

router ospf
  ospf router-id 1.1.1.1
  network 10.0.1.0/24 area 0
  network 10.0.10.0/24 area 0
  network 10.0.11.0/24 area 0
log file /var/log/quagga/ospfd.log
```

Gambar 3. 14 File r1ospfd.conf

Edit r2ospfd.conf

```
  GNU nano 4.8                                                  r2ospfd.conf
hostname r2
password zebra
enable password zebra

interface r2-eth1
 ip ospf hello-interval 1
 ip ospf dead-interval 4

interface r2-eth2
 ip ospf hello-interval 1
 ip ospf dead-interval 4

router ospf
  ospf router-id 2.2.2.2
  network 10.0.2.0/24 area 0
  network 10.0.10.0/24 area 0
  network 10.0.12.0/24 area 0
log file /var/log/quagga/ospfd.log
```

Gambar 3. 15 File r2ospfd.conf

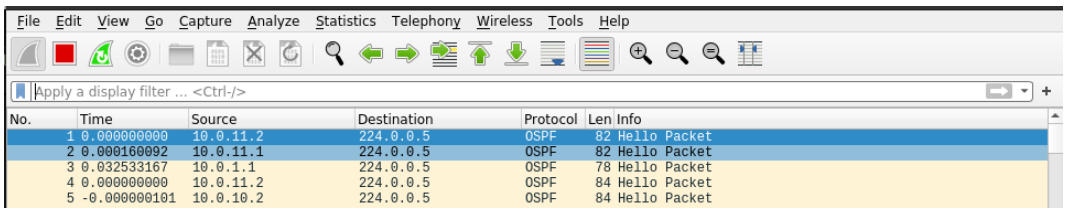
Edit r3ospfd.conf

```
  GNU nano 4.8                                                  r3ospfd.conf
hostname r3
password zebra
enable password zebra

interface r3-eth1
 ip ospf hello-interval 1
 ip ospf dead-interval 4

interface r3-eth2
 ip ospf hello-interval 1
 ip ospf dead-interval 4

router ospf
  ospf router-id 3.3.3.3
  network 10.0.3.0/24 area 0
  network 10.0.11.0/24 area 0
  network 10.0.13.0/24 area 0
log file /var/log/quagga/ospfd.log
```

Gambar 3. 16 File r3ospfd.conf

3.8 Run OSPF

Setelah konfigurasi zebra dan OSPF dilakukan, maka OSPF dapat diuji untuk melihat, apakah protokol routing OSPF sudah berjalan atau belum :



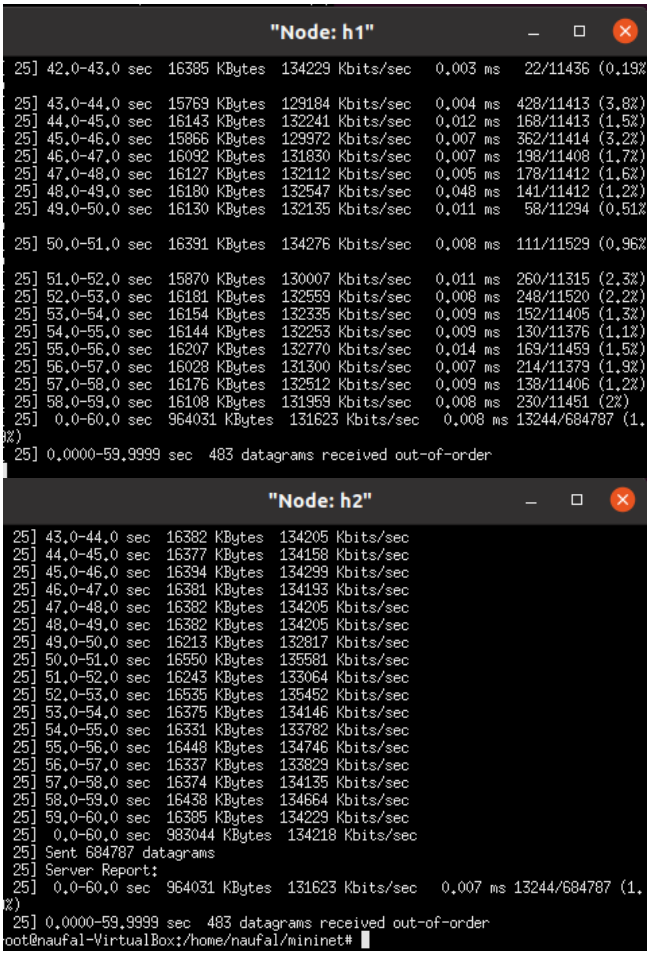
Gambar 3. 17 OSPF

3.9 Pengujian Jaringan

Pada tahap ini, dilakukan pengujian kinerja protokol routing OSPF menggunakan controller RYU dan OpenDaylight pada jaringan *Software Defined Network* (SDN). Pengujian ini dilakukan untuk mengetahui perilaku jaringan *Software Defined Network* dan bagaimana tingkat kemampuan *Software Defined Network* dalam menangani skala jaringan dengan menggunakan protokol routing OSPF serta mengukur kinerja jaringan dengan parameter *throughput*, *latency*, dan *packet loss.*

3.9.1 Pengujian Throughput pada Controller SDN

Pengujian *throughput* yang dilakukan pada mininet dengan menggunakan tool *iperf* dilakukan pada protokol UDP Buffer Size yaitu 1024Kbytes dan Bandwitch sebesar 1024Mbps. Pengujian *throughput* dilakukan untuk mengetahui banyaknya flow yang direspon oleh controller.



Gambar 3. 18 Pengujian Throughput

Seperti pada gambar 3.18 node h1 bertindak sebagai penerima packet traffic (server), sedangkan node h2 bertindak sebagai pengirim packet traffic (client). Host yang bertindak sebagai server dijalankan dengan perintah :

```
iperf -s -u -i 1 -p 5001 -w 128K -f k
```

Pada host yang bertindak sebagai client dijalankan dengan perintah :

```
iperf -c 10.0.1.100 -u -i 1 -p 5001 -w 128K -f k -b 128M -t 10
```

Keterangan :

-s : server.

-u : protokol UDP.

-c : client.

-b : setting bandwitch, data yang dikirim dari client ke server.

128M : bandwitch di setting sebanyak 128Mbps.

-t10 : waktu durasi pengujian dalam detik, yaitu 10 detik.

-t30 : waktu durasi pengujian dalam detik, yaitu 30 detik.

-t60 : waktu durasi pengujian dalam detik, yaitu 60 detik.

-i 1 : interval dalam detik antara laporan bandwitch periodik, yaitu 1.

-w : ukuran buffer size disetting.

128K : ukuran buffer size disetting sebanyak 128Kbyte.

Pada pengujian seperti gambar 3.18 dilakukan pada controller RYU dengan durasi waktu 60 detik, untuk pengujian dengan durasi durasi 10 detik dan 30 detik dilakukan dengan cara yang sama.

Begitupun untuk pengujian *throughput* pada controller OpenDaylight dilakukan dengan cara yang sama pada controller RYU dengan masing – masing waktu durasi pengujian.

3.9.2 Pengujian Latency pada Controller SDN

Pengujian *latency* yang dilakukan pada mininet dilakukan pada besar packet ICMP 1024byte. Pengujian Latency dilakukan untuk mengetahui banyaknya waktu yang dibutuhkan controller untuk memberikan respon dalam tiap detiknya.

Metode yang digunakan untuk melakukan pengecekan koneksi jaringan yaitu *PING (Packet Internet Gapher)* yang berfungsi sebagai pelacakan konektivitas antara satu komputer dengan komputer lainnya dengan mengirim sebuah pesan *ICMP (Internet Control Message Protocol).*

Gambar 3. 19 Pengujian Latency

Keterangan :

10. 0. 1. 100 : IP address dari h1 (server).

-s 1024 : menentukan ukuran payload ICMP sebesar 1024 byte.

-c : mengirim jumlah packet lalu berhenti.

- *Bytes* merupakan besaran packet yang dikirimkan, apabila kita tidak menyatakan besaaran packet yang akan dikirimkan, maka secara otomatis akan menentukan sendiri besaran paket datanya, yaitu 64 bytes pada linux.
- *Time* adalah durasi waktu yang dibutuhkan packet yang dikirim sampai ke tujuan dan waktu yang dibutuhkan oleh penerima untuk memberikan respon bahwa packet sudah diterima.
- *TTL (Time To Live)* adalah semacam penanda waktu agar packet kiriman *ping* tidak menerus terkirim. TTL menandakan bahwa packet *ping* harus berakhir dalam jangka waktu tertentu.

- Statistik adalah hasil akhir dari perintah ping yang dijalankan, pada bagian ini disebutkan berapa jumlah packet yang dikirim, jumlah packet yang diterima, dan juga presentasi packet yang hilang ditengah jalan serta berisi informasi terkait rata – rata yang dibutuhkan.

3.9.3 Pengujian Packet Loss pada Controller SDN

Pengujian *packet loss* dilakukan untuk mengetahui banyaknya packet yang hilang atau gagal yang tidak sampai pada tujuan. *Packet loss* diukur dalam persen (%). Packet dapat hilang karena disebabkan oleh *collision* dan *congestion* pada jaringan.

BAB 4
HASIL DAN PEMBAHASAN

Pada bagian ini menjelaskan hasil pengujian terhadap implementasi yang telah dilakukan beserta analisisnya. Pengujian dilakukan untuk menilai apakah seluruh kebutuhan dan analisis yang telah dispesifikasikan sebelumnya telah terpenuhi. Pengujian merupakan perbandingan performa dari controller OpenDaylight dan RYU.

4.1 Hasil Pengujian

4.1.1 Throughput

Throughput adalah jumlah data yang diterima dari pengirim dan ke penerima. *Throughput* diuji selama tiga kali pengujian dengan menggunakan tool *iperf*. Akan diperoleh masing – masing pengujian dengan waktu durasi yang berbeda dan selang waktu 1 detik pada setiap waktu durasi. Dengan pengujian menggunakan *iperf*, sehingga didapatkan nilai UDP sebagai berikut.

Table 4. 1 Nilai UDP Throughput controller Opendaylight dan RYU

Durasi Waktu	**OpenDaylight**		**RYU**	
	Paket Dikirim	**Paket Diterima**	**Paket Dikirm**	**Paket Diterima**
10 s	114132	114127	114130	114130
30 s	342395	342124	342394	342394
60 s	684784	684189	684786	684786

Setelah didapatkan nilai dari pengujian iperf, kemudian dihitung menggunakan rumus persamaan *throughput* :

$$Throughput = \frac{jumlah\ data\ yang\ diterima}{lama\ pengamatan}$$

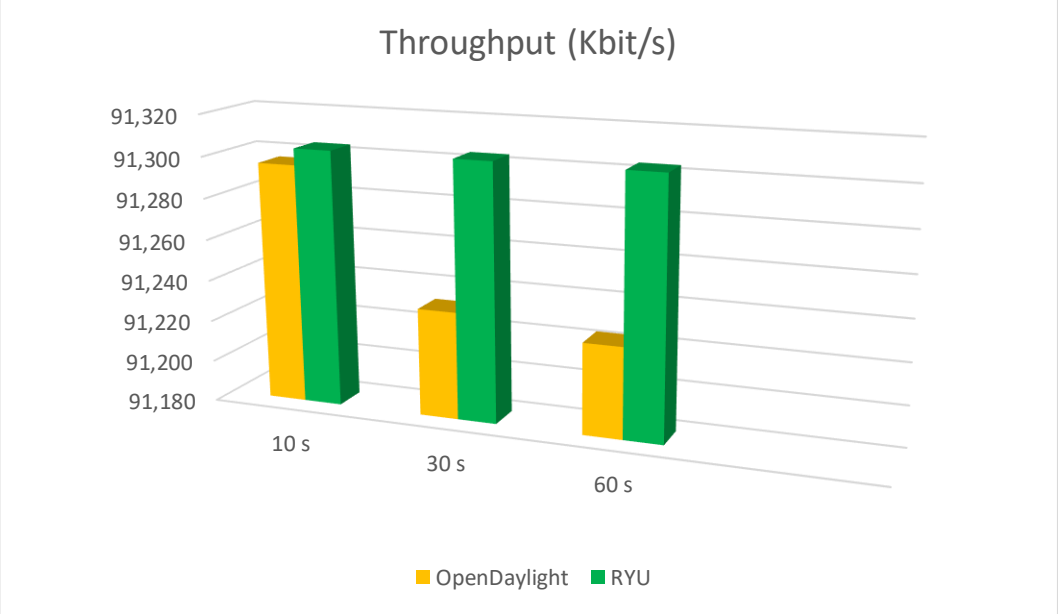
Untuk mengkonversi ke dalam bit, maka hasil pembagiannya akan dikali 8.

Pada table dibawah ini menunjukkan hasil perhitungan yang diperoleh dengan menggunakan rumus persamaan *throughput*.

Table 4. 2 Hasil Perhitungan *Throughput* controller OpenDaylight dan RYU

Durasi Waktu	**Controller**	
	OpenDaylight	**RYU**
10 s	91.296 Kbit/s	91.304 Kbit/s
30 s	91.232 Kbit/s	91.304 Kbit/s
60 s	91.224 Kbit/s	91.304 Kbit/s

Dari hasil perhitungan *throughput* dengan menggunakan rumus persamaan *throughput*, didapatkan grafik sebagai berikut.



Gambar 4. 1 Grafik *Throughput* Controller OpenDaylight dan RYU

Pada grafik 4.1 merupakan hasil dari perbandingan *throughput* controller OpenDaylight dan RYU dimana pada pengujian ini diberikan ukuran buffer size

yaitu 1024Kbyte dan diberikan ukuran bandwitdh 1024Mbyte dengan durasi waktu yang berbeda.

Pada tabel 4.2 dapat dilihat nilai throughput pada controller OpenDaylight dan RYU dan. Pada durasi waktu 10 detik, throughput pada controller OpenDaylight 91.296 Kbit/s dan controller RYU 91.304 Kbit/s, durasi waktu 30 detik throughput controller OpenDaylight 91.232 Kbit/s dan controller RYU 91.304 Kbit/s, sedangkan pada durasi waktu 60 detik throughput dari controller OpenDaylight 91.224 Kbit/s dan controller RYU 91.304 Kbit/s.

Berdasarkan grafik yang diperoleh dari masing-masing durasi waktu untuk parameter *throughput* menunjukkan bahwa pada controller RYU, lebih baik dibandingkan dengan controller OpenDaylight. Hal ini terjadi karena adanya perbedaan skalabilitas dari kedua controller, dimana pada controller OpenDaylight memiliki skalabilitas yang sangat terbatas, sedangkan skalabilitas pada controller RYU sangat baik. Selain dari perbedaan skalabilitas, perbedaan dari controller OpenDaylight dan controller RYU adalah realibility. Controller OpenDaylight memiliki realibility yang terbatas sedangkan controller RYU memiliki realibilty yang baik. Maka dari itu, hasil *throughput* dari controller RYU lebih tinggi jika dibandingkan dengan controller OpenDaylight.

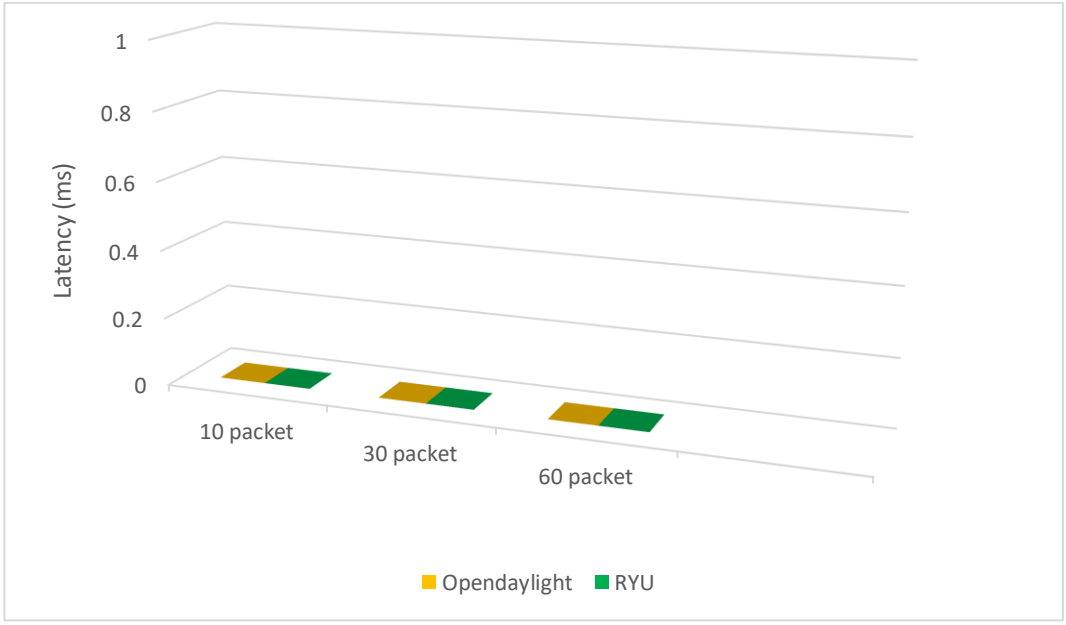
4.4.2 Latency

Nilai rata-rata *latency* yang didapatkan pada setiap jumlah packet yang dikrim menggunakan *ICMP* dengan melakukan ping kedua host. Nilai rata-rata ini memberikan indikasi umum tentang waktu yang dibutuhkan oleh paket data untuk melakukan perjalanan dalam jaringan, dengan mempertimbangkan berbagai kondisi jaringan yang mungkin berfluktuasi. Menghitung rata-rata latency melibatkan pengiriman sejumlah paket data dan pengukuran waktu tempuh masing-masing paket, kemudian menghitung nilai rata-ratanya. Pengukuran latency yang akurat sangat penting untuk memahami responsivitas dan efisiensi jaringan, terutama untuk aplikasi yang membutuhkan waktu respons cepat seperti streaming video, game online, dan aplikasi real-time lainnya. Untuk mendapatkan gambaran yang lebih jelas mengenai performa jaringan, sering kali dilakukan pengukuran rata-rata latency Nilai rata-rata *latency* dapat dilihat pada table berikut.

Table 4. 3 Hasil ping controller OpenDaylight dadan RYU

Jumlah Packet yang Dikirim	**Controller**	
	OpenDaylight	**RYU**
10 packet	0,095 ms	0,078 ms
30 packet	0,116 ms	0,106 ms
60 packet	0,135 ms	0,114 ms

Dari hasil perhitungan yang telah dilakukan menggunakan ICMP Ping dapat dilihat pada grafik dibawah ini.



Gambar 4. 2 Grafik Latency Controller OpenDaylight dan RYU

Dari data yang terlihat pada tabel 4.3 dan grafik pada grafik 4.2 di atas, dapat dilihat bahwa kedua controller mengalami peningkatan *latency* ketika jumlah packet yang dikirim bertambah. Semakin bertambah jumlah packet yang dikirim, maka *latency*-nya juga akan semakin meningkat.

Nilai *latency* yang lebih kecil berarti lebih baik. Dapat disimpulkan bahwa rata-rata *latency* menggunakan controller RYU lebih baik dibanding menggunakan controller OpenDaylight. Jika kita menggunakan referensi standarisasi TIPHON, maka controller RYU lebih baik dari controller OpenDaylight, karena *latency* dari controller RYU lebih kecil dibandingkan dengan controller OpenDaylight.

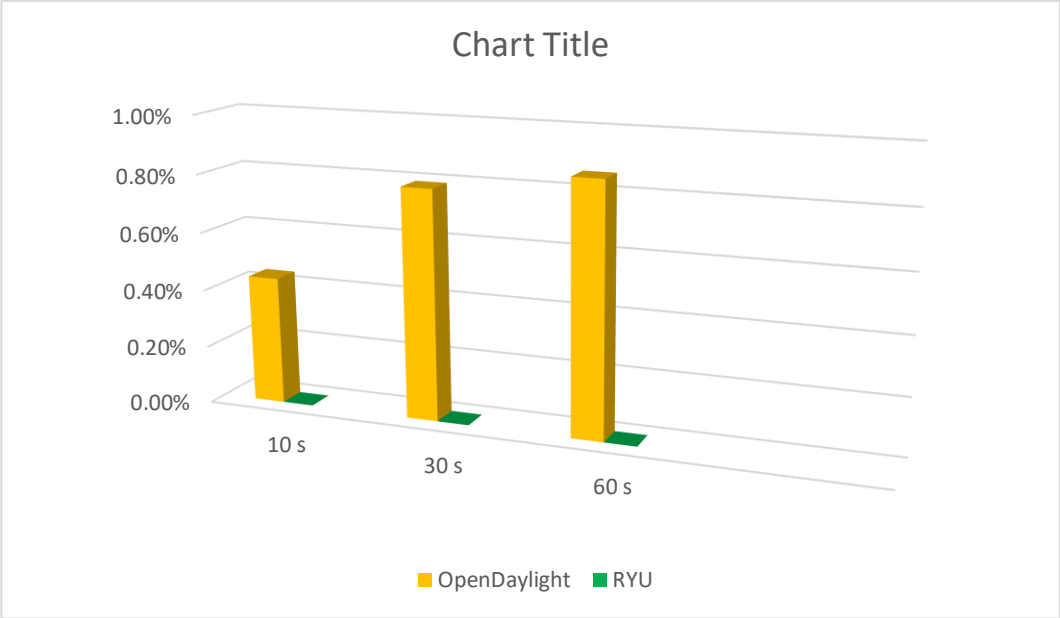
4.4.3 Packet Loss

Packet loss terjadi ketika satu atau lebih paket data yang dikirim melalui jaringan tidak mencapai tujuan yang ditentukan. Ini bisa disebabkan oleh berbagai faktor, seperti kemacetan jaringan, kesalahan perangkat keras, atau masalah dalam routing. Untuk memahami dan mengukur performa jaringan secara komprehensif, penting untuk melakukan pengujian *packet loss*. Pengukuran ini membantu mengidentifikasi bagian jaringan yang bermasalah dan dapat memberikan wawasan tentang bagaimana meningkatkan kualitas dan keandalan jaringan. Pengujian *packet loss* UDP menggunakan tool *iperf* dengan mengirimkan paket UDP, masing-masing pengujian dengan mengirimkan paket datagram sebanyak-banyaknya dalam durasi waktu yang berbeda beda dengan selang waktu 1 detik dengan ukuran *buffer size* 1024KByte. Sehingga, didapatkan nilai *packet loss* sebagai berikut.

Table 4. 4 Hasil Perhitungan Packet Loss controller Opendaylight dan RYU

Jumlah Packet yang Dikirim	**Controller**	
	OpenDaylight	**RYU**
10 s	0,44%	0%
30 s	0,79%	0%
60 s	0,86%	0%

Dari hasil perhitungan *packet loss* menggunakan tool *iperf*, didapatkan grafik sebagai berikut.



Gambar 4. 3 Grafik Packet Loss Controller OpenDaylight dan RYU

Berdasarkan grafik yang diperoleh dari masing-masing durasi waktu untuk parameter *packet loss*, menunjukka bahwa controller RYU lebih baik dibandingkan dengan controller OpenDaylight. Hal ini disebabkan karena, controller OpenDaylight lebih tinggi *packet lossnya* atau paket dikirim tidak sampai pada tempat tujuannya.

Jika kita menggunakan referensi standarisasi TIPHON maka, controller RYU pada durasi waktu 10, 30, 60 detik sama – sama memiliki nilai *packet loss* 0%. Pada durasi waktu 10 detik, controller OpenDaylight memiliki nilai *packet loss* 0,44%. Pada durasi 30 detik controller OpenDaylight memiliki nilai *packet loss* 0,79%. Sedangkan pada durasi waktu 60 detik, nilai *packet loss* dari controller OpenDaylight 0,86%. Maka dapat disimpulkan bahwa controller RYU lebih baik kinerjanya dibandingkan dengan controller OpenDaylight.

4.1 Pembahasan

Dari hasil pengujian yang telah dilakukan, dapat disimpulkan bahwa RYU memiliki kinerja yang lebih baik dibandingkan OpenDaylight dalam hal *throughput*, *latency*, dan *packet loss*. Keunggulan ini dapat dikaitkan dengan arsitektur dan mekanisme internal RYU yang lebih efisien dalam menangani protokol routing OSPF.

Throughput yang lebih tinggi pada RYU menunjukkan bahwa controller ini mampu menangani jumlah data yang lebih besar dalam waktu yang sama dibandingkan OpenDaylight. Kemampuan untuk mentransmisikan data dengan cepat sangat penting dalam jaringan dengan lalu lintas tinggi. Mekanisme pengelolaan jalur data yang efisien dan optimisasi pengolahan data pada RYU berkontribusi pada *throughput* yang lebih tinggi.

RYU menunjukkan performa *latency* yang lebih rendah dibandingkan OpenDaylight. *Latency* yang lebih rendah berarti waktu perjalanan paket dari sumber ke tujuan dan kembali lagi lebih singkat, yang penting untuk aplikasi yang membutuhkan respons waktu nyata. Efisiensi RYU dalam mengelola jalur transmisi dan memproses paket data secara cepat mungkin menjadi faktor utama dalam pencapaian *latency* yang lebih rendah ini.

Packet loss yang lebih rendah pada RYU menunjukkan bahwa controller ini lebih andal dalam menjaga integritas data selama transmisi. Kehilangan paket yang minimal sangat penting untuk menjaga kualitas layanan, terutama untuk aplikasi yang sensitif terhadap kehilangan data seperti video streaming dan VoIP. Algoritma pemulihan dan pengelolaan paket yang efektif pada RYU mungkin menjadi faktor utama dalam pencapaian tingkat *packet loss* yang lebih rendah ini.

BAB 5
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kesimpulan dari seluruh proses yang dilakukan dan hasil pembahasan penelitian yang telah dilakukan yaitu dapat menghasilkan :

1. Hasil analisis menunjukkan bahwa implementasi routing protokol OSPF di lingkungan SDN menggunakan controller Ryu dan OpenDaylight mampu mengelola dan mengoptimalkan penggunaan sumber daya jaringan dengan lebih efisien.

2. Performa jaringan SDN dengan routing protokol OSPF menggunakan controller RYU menunjukkan throughput yang lebih tinggi, latensi yang lebih rendah, dan tingkat packet loss yang lebih rendah dibandingkan dengan OpenDaylight. Ini mengindikasikan bahwa RYU memiliki kinerja yang lebih baik dalam hal efisiensi transfer data, waktu respons, dan reliabilitas jaringan.

3. Pada penelitian ini, implementasi controller OpenDaylight dan Ryu telah berhasil dilakukan dan menghasilkan performa yang berbeda. Dalam hal ini controller RYU memiliki performa lebih baik daripada controller OpenDaylight dalam parameter *throughtput, latency, dan packet loss*. *RYU* mampu memberikan *throughput* yang lebih besar dibandingkan dengan controller OpenDaylight, sedangkan kinerja *latency* yang disajikan dalam jumlah respon/detik, controller RYU memberikan respon yang lebih baik dibandingkan dengan controller OpenDaylight.

5.2 Saran

1. Melakukan penelitian lebih lanjut untuk mengoptimalkan algoritma dan strategi OSPF yang sesuai dengan karakteristik SDN secara spesifik.

2. Mengembangkan studi kasus atau simulasi yang lebih komprehensif untuk memperoleh wawasan yang lebih dalam tentang interaksi antara OSPF, controller SDN, dan elemen jaringan lainnya.
3. Menerapkan hasil penelitian ini dalam skala yang lebih luas untuk menguji validitas dan keunggulan implementasi OSPF di SDN pada berbagai jenis jaringan dan skenario penggunaan.

DAFTAR PUSTAKA

Abidin, N. Z. (2021). Analisis Performansi Controller POX Dan RYU Pada Jaringan Software Defined Network Dengan Protokol Spanning Tree. *Repository.Uinjkt.Ac.Id*, 118. https://repository.uinjkt.ac.id/dspace/handle/123456789/56384

Adrian, R. (2017). Optimasi Cost pada Open Shortest Path First di Jaringan Software Defined-Network. *Techno.Com*, *16*(4), 421–434. https://doi.org/10.33633/tc.v16i4.1532

Ainy, M. (2017). Routing Interior dan Eksterior. *Fakultas Komputer*.

Attamimi, I., Yahya, W., & Hanafi, M. H. (2017). Analisis Perbandingan Algoritma Floyd-Warshall dan Dijkstra untuk Menentukan Jalur Terpendek Pada Jaringan Openflow. *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer (J-PTIIK)*, *1*(12), 1842–1849.

Cintasari, E. P. (2018). Analisis Kinerja Jaringan Software Defined Network (SDN) Dengan Protokol OpenFlow pada Mininet. *Repository.Uinjkt.Ac.Id*. http://repository.uinjkt.ac.id/dspace/handle/123456789/53507

Dwi Rahmawan, A., & Risqiwati, D. (2020). Analisa Performansi Controller Pada Arsitektur Jaringan Software Defined Network (SDN). *REPOSITOR*, *2*(12), 1727–1738.

Edgar, R., Hanuranto, A. T., & Mentari, O. (2019). *Perancangan Dan Analisis Sistem Pada Kontroler Pox, Ryu, Dan Opendaylight Pada Software Defined Network*. *6*(2), 4433–4441.

Indriani Lestariningati, S. (2018). Analisis Kinerja Arsitektur Software-Defined Network Berbasis OpenDaylight Controller. *Jurnal Teknik Komputer Unikom-Komputika*, *7*(1), 194–200.

Irmawati, A., Irawati, I. D., & Hariyani, Y. S. (2017). Implementasi Protokol Routing Ospf Pada Software Defined Network Berbasis Routeflow. *E-Proceeding of Apllied Science*, *3*(2), 1067–1074.

Iryani, N., Ramadhani, A. D., & Sari, M. K. (2021). Analisis Performansi Routing OSPF menggunakan RYU Controller dan POX Controller pada Software Defined Networking. *Jurnal Telekomunikasi Dan Komputer*, *11*(1), 73. https://doi.org/10.22441/incomtech.v11i1.10187

Kementrian Kesehatan Republik Indonesia. (2018). UNIVERSITAS SUMATERA UTARA Poliklinik UNIVERSITAS SUMATERA UTARA. *Jurnal Pembangunan Wilayah & Kota*, *1*(3), 82–91.

Khattak, Z. K., Awais, M., & Departemen, I. (2014). *Evaluasi Kinerja OpenDaylight SDN Pengendali*.

Khoerul, A., & Ronald, A. (2017). Analisis Performa Jaringan Software Defined Network Berdasarkan Penggunaan Cost Pada Protokol Routing Open Shotest Path First. *Citee*, 1–8.

Stmik-amik-riau, L. K., Muzawi, R., & Hardianto, R. (2016). *Perancangan Server Dan Analisis Quality of Service (QoS) Jaringan Diskless PXE Linux Pada*. *1*(1).

Sudiyatmoko, A. R., Hertiana, S. N., & Negara, R. M. (2016). Analisis Performansi Perutingan Link State Menggunakan Algoritma Djikstra Pada Platform Software Defined Network (SDN). *JURNAL INFOTEL - Informatika Telekomunikasi Elektronika*, *8*(1), 40. https://doi.org/10.20895/infotel.v8i1.50

Sulfiana, A. (2019). Implementasi dan Analisis Kinerja Software Defined Network Controller Floodlight dan Onos. *Estuarine, Coastal and Shelf Science*, *2020*(1), 473–484.

Thomas, E. E., Palit, H., & Noertjahyana, A. (2018). Aplikasi Manajemen Jaringan Berbasis Software Defined Networking. *Jurnal Infra Petra*, *031*. http://publication.petra.ac.id/index.php/teknik-informatika/article/view/6384

Time, R., & Time, R. (2020). *1. Perkenalan*. *0123456789*.

Ummah, I. (2016). Perancangan Simulasi Jaringan Virtual Berbasis Software-Define Networking. *Indonesian Journal on Computing (Indo-JC)*, *1*(1), 95–106. https://doi.org/10.21108/indojc.2016.1.1.20

Utami, P. R. (2020). Analisis Perbandingan Quality of Service Jaringan Internet Berbasis Wireless Pada Layanan Internet Service Provider (Isp) Indihome Dan First Media. *Jurnal Ilmiah Teknologi Dan Rekayasa*, *25*(2), 125–137. https://doi.org/10.35760/tr.2020.v25i2.2723

Zhu, L., Karim, M. M., Sharif, K., Li, F., Du, X., & Guizani, M. (2019). *SDN Controllers: Benchmarking & Performance Evaluation*. 1–14. http://arxiv.org/abs/1902.04491

홍종욱. (2019). 3월 1일의 밤은 대한민국의 봄이었다 — 권보드래, ?3월 1일의 밤: 폭력의 세기에 꾸는 평화의 꿈? (돌베개, 2019) —. *Concept and Communication*, *null*(23), 301–316. https://doi.org/10.15797/concom.2019..23.009

LAMPIRAN

Lampiran 1 Script OSPF dan RYU pada Mininet

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import Node, RemoteController, OVSSwitch
from mininet.log import setLogLevel, info
from mininet.cli import CLI
import time
import os

class LinuxRouter(Node):
 "Node dengan IP forwarding diaktifkan."

 def config(self, **params):
 super(LinuxRouter, self).config(**params)
 # Aktifkan forwarding pada router
 self.cmd('sysctl net.ipv4.ip_forward=1')

 def terminate(self):
 self.cmd('sysctl net.ipv4.ip_forward=0')
 super(LinuxRouter, self).terminate()

class NetworkTopo(Topo):
 "Sebuah LinuxRouter yang menghubungkan tiga subnet IP"

 def build(self, **_opts):
 # Alamat IP untuk antarmuka router
 r1_eth1_ip = '10.0.1.1/24' # r1 ke h1
 r2_eth1_ip = '10.0.2.1/24' # r2 ke h2
 r3_eth1_ip = '10.0.3.1/24' # r3 ke h3

 r1_eth2_ip = '10.0.10.1/24' # r1 ke r2
 r2_eth2_ip = '10.0.10.2/24' # r2 ke r1

 r1_eth3_ip = '10.0.11.1/24' # r1 ke r3
 r3_eth2_ip = '10.0.11.2/24' # r3 ke r1

 r2_eth3_ip = '10.0.12.1/24' # r2 ke r3
 r3_eth3_ip = '10.0.12.2/24' # r3 ke r2

 # Tambahan alamat IP untuk antarmuka switch ke router
 s1_eth1_ip = '10.0.1.2/24' # s1 ke r1
 s2_eth1_ip = '10.0.2.2/24' # s2 ke r2
 s3_eth1_ip = '10.0.3.2/24' # s3 ke r3

 # Menambahkan router
 r1 = self.addHost('r1', cls=LinuxRouter, ip=r1_eth1_ip)
```

```
 r2 = self.addHost('r2', cls=LinuxRouter, ip=r2_eth1_ip)
 r3 = self.addHost('r3', cls=LinuxRouter, ip=r3_eth1_ip)

 # Menambahkan host
 h1 = self.addHost('h1', ip='10.0.1.100/24', defaultRoute='via
10.0.1.1')
 h2 = self.addHost('h2', ip='10.0.2.100/24', defaultRoute='via
10.0.2.1')
 h3 = self.addHost('h3', ip='10.0.3.100/24', defaultRoute='via
10.0.3.1')

 # Menambahkan switch
 s1 = self.addSwitch('s1', cls=OVSSwitch,
protocols='OpenFlow13')
 s2 = self.addSwitch('s2', cls=OVSSwitch,
protocols='OpenFlow13')
 s3 = self.addSwitch('s3', cls=OVSSwitch,
protocols='OpenFlow13')

 # Membuat link dengan alamat IP untuk antarmuka switch ke
router
 self.addLink(s1, r1, intfName1='s1-eth1', params1={'ip':
s1_eth1_ip}, intfName2='r1-eth1', params2={'ip': r1_eth1_ip})
# Switch s1 ke Router r1
 self.addLink(s2, r2, intfName1='s2-eth1', params1={'ip':
s2_eth1_ip}, intfName2='r2-eth1', params2={'ip': r2_eth1_ip})
# Switch s2 ke Router r2
 self.addLink(s3, r3, intfName1='s3-eth1', params1={'ip':
s3_eth1_ip}, intfName2='r3-eth1', params2={'ip': r3_eth1_ip})
# Switch s3 ke Router r3

 self.addLink(r1, h1, intfName1='r1-eth4', params1={'ip':
'10.0.1.1/24'}) # Router r1 ke Host h1
 self.addLink(r2, h2, intfName1='r2-eth4', params1={'ip':
'10.0.2.1/24'}) # Router r2 ke Host h2
 self.addLink(r3, h3, intfName1='r3-eth4', params1={'ip':
'10.0.3.1/24'}) # Router r3 ke Host h3

 self.addLink(r1, r2, intfName1='r1-eth2', intfName2='r2-
eth2', params1={'ip': r1_eth2_ip}, params2={'ip': r2_eth2_ip})
 self.addLink(r1, r3, intfName1='r1-eth3', intfName2='r3-
eth2', params1={'ip': r1_eth3_ip}, params2={'ip': r3_eth2_ip})
 self.addLink(r2, r3, intfName1='r2-eth3', intfName2='r3-
eth3', params1={'ip': r2_eth3_ip}, params2={'ip': r3_eth3_ip})

 # Link tambahan untuk menghubungkan switch
 self.addLink(s1, s2)
 self.addLink(s2, s3)
```

```
 def configure_control_plane(self, net):
 "Konfigurasi zebra dan ospf untuk setiap router"
 r1 = net.getNodeByName('r1')
 r2 = net.getNodeByName('r2')
 r3 = net.getNodeByName('r3')

 r1.cmd('zebra -f /usr/local/etc/r1zebra.conf -d -z
~/r1zebra.api -i ~/r1zebra.interface')
 time.sleep(1)
 r2.cmd('zebra -f /usr/local/etc/r2zebra.conf -d -z
~/r2zebra.api -i ~/r2zebra.interface')
 r3.cmd('zebra -f /usr/local/etc/r3zebra.conf -d -z
~/r3zebra.api -i ~/r3zebra.interface')

 r1.cmd('ospfd -f /usr/local/etc/r1ospfd.conf -d -z
~/r1zebra.api -i ~/r1ospfd.interface')
 r2.cmd('ospfd -f /usr/local/etc/r2ospfd.conf -d -z
~/r2zebra.api -i ~/r2ospfd.interface')
 r3.cmd('ospfd -f /usr/local/etc/r3ospfd.conf -d -z
~/r3zebra.api -i ~/r3ospfd.interface')

def run():
 "Uji router Linux"
 topo = NetworkTopo()
 net = Mininet(controller=RemoteController('ryu',
ip='127.0.0.1', port=6633), topo=topo, switch=OVSSwitch)
 net.start()
 info('*** Tabel Routing pada Router:\n')

 topo.configure_control_plane(net)

 # Verifikasi koneksi controller
 for switch in net.switches:
 switch.cmd('ovs-vsctl set-controller %s tcp:127.0.0.1:6633' %
switch.name)

 # Tunggu beberapa detik agar topologi muncul di Ryu
 time.sleep(10)

 # Jalankan iperf untuk menghasilkan trafik UDP
 info('*** Menghasilkan trafik UDP:\n')
 h1, h2, h3 = net.get('h1', 'h2', 'h3')
 h1.cmd('iperf -s -u -i 1 -p 5001 -w 128K -f k &')
 time.sleep(1) # Tunggu server untuk memulai
 h2.cmd('iperf -c 10.0.1.100 -u -i 1 -p 5001 -w 128K -f k -b
128M &')

 # Tambahkan rute default untuk h3
 h3.cmd('ip route add default via 10.0.3.1')
```

```
 CLI(net)

 net.stop()

 os.system("killall -9 ospfd zebra")
 os.system("rm -f *api*")
 os.system("rm -f *interface*")

if __name__ == '__main__':
 setLogLevel('info')
 run()
```

Lampiran 2 Script OSPF dan OpenDaylight pada Mininet

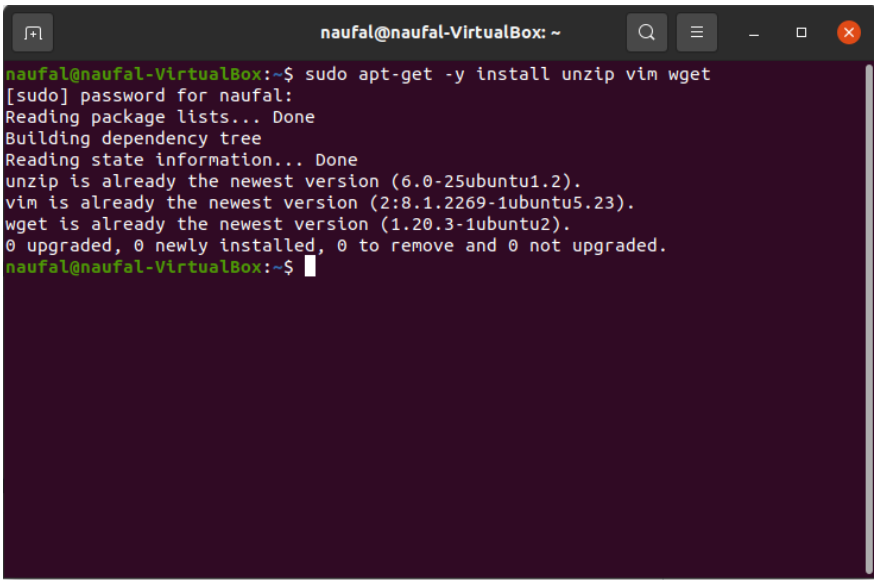
```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import Node, RemoteController, OVSSwitch
from mininet.log import setLogLevel, info
from mininet.cli import CLI
import time
import os

class LinuxRouter(Node):
 "Node dengan IP forwarding diaktifkan."

 def config(self, **params):
 super(LinuxRouter, self).config(**params)
 # Aktifkan forwarding pada router
 self.cmd('sysctl net.ipv4.ip_forward=1')

 def terminate(self):
 self.cmd('sysctl net.ipv4.ip_forward=0')
 super(LinuxRouter, self).terminate()

class NetworkTopo(Topo):
 "Sebuah LinuxRouter yang menghubungkan tiga subnet IP"

 def build(self, **_opts):
 # Alamat IP untuk antarmuka router
 r1_eth1_ip = '10.0.1.1/24' # r1 ke h1
 r2_eth1_ip = '10.0.2.1/24' # r2 ke h2
 r3_eth1_ip = '10.0.3.1/24' # r3 ke h3

 r1_eth2_ip = '10.0.10.1/24' # r1 ke r2
```

```
 r2_eth2_ip = '10.0.10.2/24' # r2 ke r1

 r1_eth3_ip = '10.0.11.1/24' # r1 ke r3
 r3_eth2_ip = '10.0.11.2/24' # r3 ke r1

 r2_eth3_ip = '10.0.12.1/24' # r2 ke r3
 r3_eth3_ip = '10.0.12.2/24' # r3 ke r2

 # Tambahan alamat IP untuk antarmuka switch ke router
 s1_eth1_ip = '10.0.1.2/24' # s1 ke r1
 s2_eth1_ip = '10.0.2.2/24' # s2 ke r2
 s3_eth1_ip = '10.0.3.2/24' # s3 ke r3

 # Menambahkan router
 r1 = self.addHost('r1', cls=LinuxRouter, ip=r1_eth1_ip)
 r2 = self.addHost('r2', cls=LinuxRouter, ip=r2_eth1_ip)
 r3 = self.addHost('r3', cls=LinuxRouter, ip=r3_eth1_ip)

 # Menambahkan host
 h1 = self.addHost('h1', ip='10.0.1.100/24', defaultRoute='via
10.0.1.1')
 h2 = self.addHost('h2', ip='10.0.2.100/24', defaultRoute='via
10.0.2.1')
 h3 = self.addHost('h3', ip='10.0.3.100/24', defaultRoute='via
10.0.3.1')

 # Menambahkan switch
 s1 = self.addSwitch('s1', cls=OVSSwitch,
protocols='OpenFlow13')
 s2 = self.addSwitch('s2', cls=OVSSwitch,
protocols='OpenFlow13')
 s3 = self.addSwitch('s3', cls=OVSSwitch,
protocols='OpenFlow13')

 # Membuat link dengan alamat IP untuk antarmuka switch ke
router
 self.addLink(s1, r1, intfName1='s1-eth1', params1={'ip':
s1_eth1_ip}, intfName2='r1-eth1', params2={'ip': r1_eth1_ip})
# Switch s1 ke Router r1
 self.addLink(s2, r2, intfName1='s2-eth1', params1={'ip':
s2_eth1_ip}, intfName2='r2-eth1', params2={'ip': r2_eth1_ip})
# Switch s2 ke Router r2
 self.addLink(s3, r3, intfName1='s3-eth1', params1={'ip':
s3_eth1_ip}, intfName2='r3-eth1', params2={'ip': r3_eth1_ip})
# Switch s3 ke Router r3

 self.addLink(r1, h1, intfName1='r1-eth4', params1={'ip':
'10.0.1.1/24'}) # Router r1 ke Host h1
```

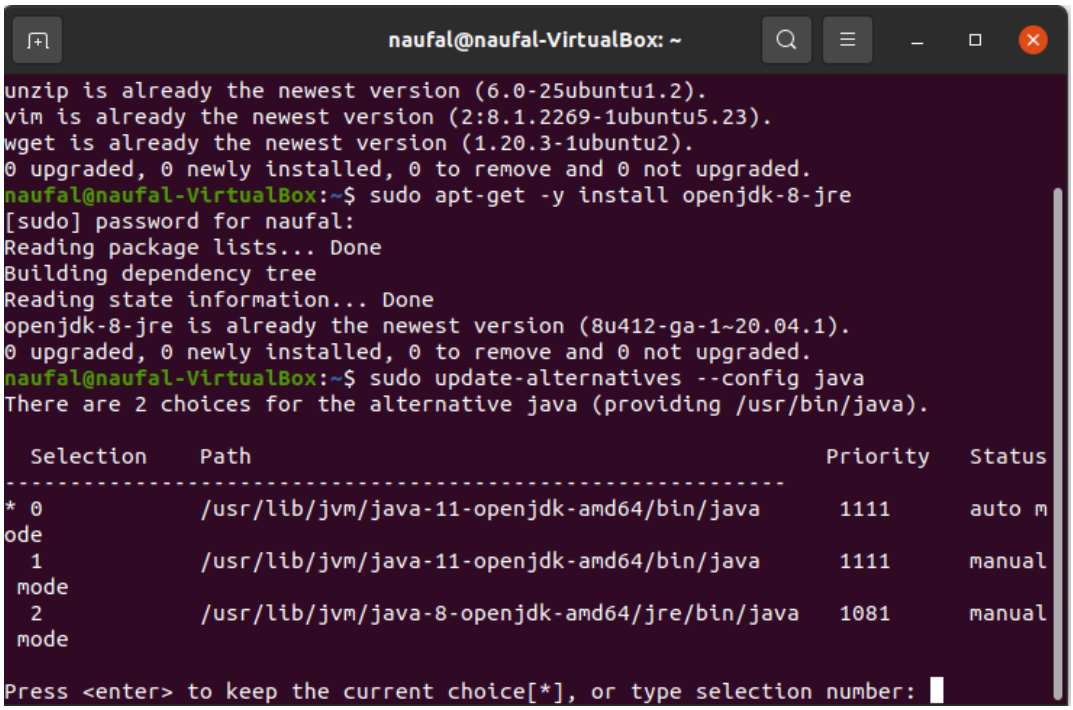
```
 self.addLink(r2, h2, intfName1='r2-eth4', params1={'ip':
'10.0.2.1/24'}) # Router r2 ke Host h2
 self.addLink(r3, h3, intfName1='r3-eth4', params1={'ip':
'10.0.3.1/24'}) # Router r3 ke Host h3

 self.addLink(r1, r2, intfName1='r1-eth2', intfName2='r2-
eth2', params1={'ip': r1_eth2_ip}, params2={'ip': r2_eth2_ip})
 self.addLink(r1, r3, intfName1='r1-eth3', intfName2='r3-
eth2', params1={'ip': r1_eth3_ip}, params2={'ip': r3_eth2_ip})
 self.addLink(r2, r3, intfName1='r2-eth3', intfName2='r3-
eth3', params1={'ip': r2_eth3_ip}, params2={'ip': r3_eth3_ip})

 # Link tambahan untuk menghubungkan switch
 self.addLink(s1, s2)
 self.addLink(s2, s3)

 def configure_control_plane(self, net):
 "Konfigurasi zebra dan ospf untuk setiap router"
 r1 = net.getNodeByName('r1')
 r2 = net.getNodeByName('r2')
 r3 = net.getNodeByName('r3')

 r1.cmd('zebra -f /usr/local/etc/r1zebra.conf -d -z
~/r1zebra.api -i ~/r1zebra.interface')
 time.sleep(1)
 r2.cmd('zebra -f /usr/local/etc/r2zebra.conf -d -z
~/r2zebra.api -i ~/r2zebra.interface')
 r3.cmd('zebra -f /usr/local/etc/r3zebra.conf -d -z
~/r3zebra.api -i ~/r3zebra.interface')

 r1.cmd('ospfd -f /usr/local/etc/r1ospfd.conf -d -z
~/r1zebra.api -i ~/r1ospfd.interface')
 r2.cmd('ospfd -f /usr/local/etc/r2ospfd.conf -d -z
~/r2zebra.api -i ~/r2ospfd.interface')
 r3.cmd('ospfd -f /usr/local/etc/r3ospfd.conf -d -z
~/r3zebra.api -i ~/r3ospfd.interface')

def run():
 "Uji router Linux"
 topo = NetworkTopo()
 net = Mininet(controller=RemoteController('odl',
ip='127.0.0.1', port=6653), topo=topo, switch=OVSSwitch)
 net.start()
 info('*** Tabel Routing pada Router:\n')

 topo.configure_control_plane(net)

 # Verifikasi koneksi controller
 for switch in net.switches:
```

```
 switch.cmd('ovs-vsctl set-controller %s tcp:127.0.0.1:6653' %
switch.name)

 # Tunggu beberapa detik agar topologi muncul di OpenDaylight
 time.sleep(10)

 # Jalankan iperf untuk menghasilkan trafik UDP
 info('*** Menghasilkan trafik UDP:\n')
 h1, h2, h3 = net.get('h1', 'h2', 'h3')
 h1.cmd('iperf -s -u -i 1 -p 5001 -w 128K -f k &')
 time.sleep(1) # Tunggu server untuk memulai
 h2.cmd('iperf -c 10.0.1.100 -u -i 1 -p 5001 -w 128K -f k -b
128M &')

 # Tambahkan rute default untuk h3
 h3.cmd('ip route add default via 10.0.3.1')

 CLI(net)

 net.stop()

 os.system("killall -9 ospfd zebra")
 os.system("rm -f *api*")
 os.system("rm -f *interface*")

if __name__ == '__main__':
 setLogLevel('info')
 run()
```

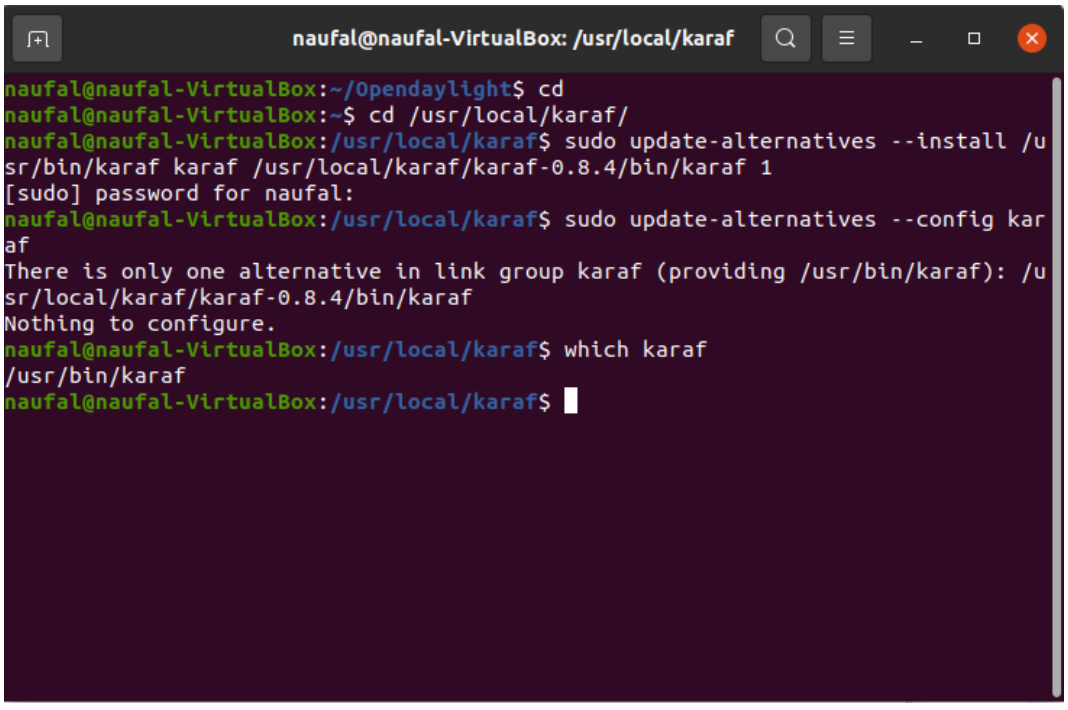
Lampiran 3 Tampilan Sistem



Lampiran 3.1 Instal Depedency OpenDaylight



Lampiran 3.2 Instalasi Jre



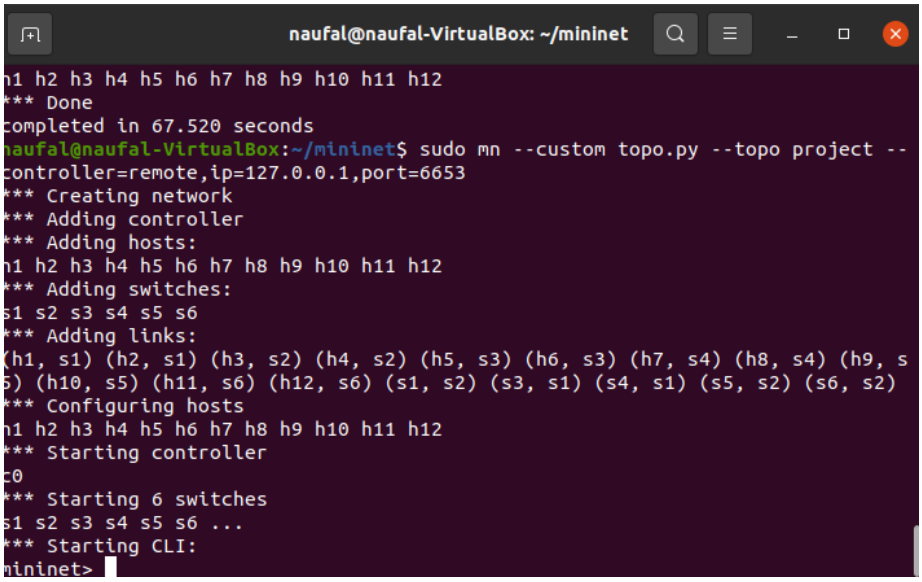
Lampiran 3.3 Instalasi Karaf atau OpenDaylight

```
naufal@naufal-VirtualBox:~/mininet$ sudo mn --test pingall
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 5.491 seconds
naufal@naufal-VirtualBox:~/mininet$
```

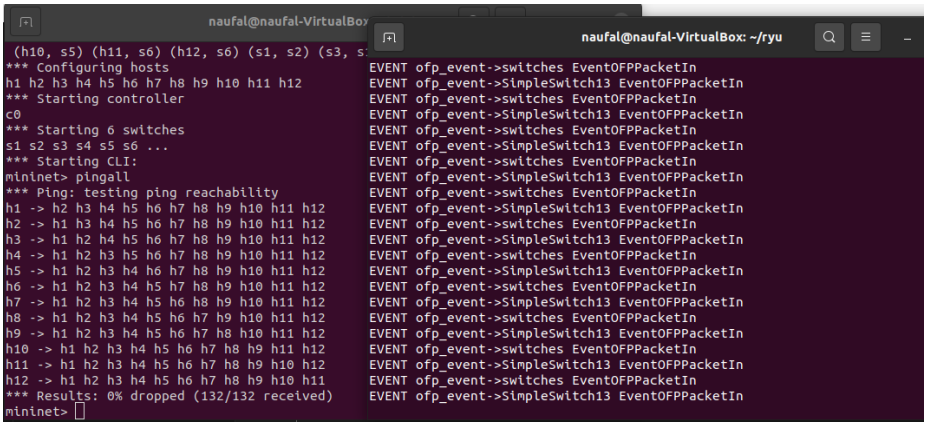
Lampiran 3.4 Instalasi Mininet

```
naufal@naufal-VirtualBox: ~/ryu
naufal@naufal-VirtualBox:~$ cd ryu
naufal@naufal-VirtualBox:~/ryu$ ryu-manager
loading app ryu.controller.ofp_handler
instantiating app ryu.controller.ofp_handler of OFPHandler
hub: uncaught exception: Traceback (most recent call last):
  File "/home/naufal/.local/lib/python3.8/site-packages/ryu/lib/hub.py", line 60
, in _launch
    return func(*args, **kwargs)
  File "/home/naufal/.local/lib/python3.8/site-packages/ryu/controller/controlle
r.py", line 204, in server_loop
    server = StreamServer((CONF.ofp_listen_host,
  File "/home/naufal/.local/lib/python3.8/site-packages/ryu/lib/hub.py", line 12
7, in __init__
    self.server = eventlet.listen(listen_info)
  File "/home/naufal/.local/lib/python3.8/site-packages/eventlet/convenience.py"
, line 78, in listen
    sock.bind(addr)
OSError: [Errno 98] Address already in use

hub: uncaught exception: Traceback (most recent call last):
  File "/home/naufal/.local/lib/python3.8/site-packages/ryu/lib/hub.py", line 60
, in _launch
    return func(*args, **kwargs)
  File "/home/naufal/.local/lib/python3.8/site-packages/ryu/controller/controlle
```

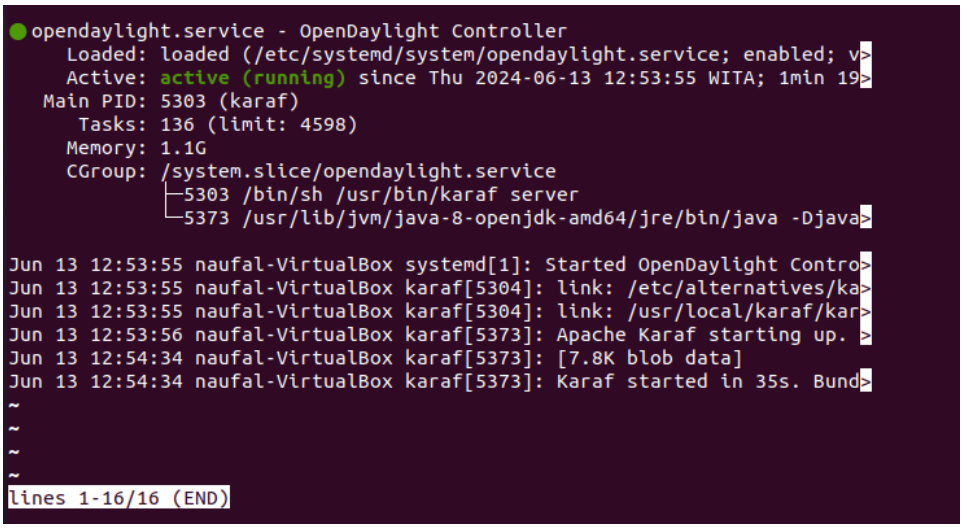
Lampiran 3.5 Instalasi RYU



Lampiran 3.6 Intergrasi Mininet dan OpenDaylight



Lampiran 3.7 Intergrasi RYU dan Mininet



Lampiran 3.8 Mengaktifkan OpenDaylight Service

```
naufal@naufal-VirtualBox:/usr/local/karaf$ sudo -E karaf
link: /etc/alternatives/karaf
link: /usr/local/karaf/karaf-0.8.4/bin/karaf
Apache Karaf starting up. Press Enter to open the shell now...
100% [========================================================================]
Karaf started in 43s. Bundle stats: 419 active, 420 total

    ________                ________                .__  .__       .__   __
    \_____  \ ______  ____  ____ \_____  \ ______ ___.__.|  | |__| ____ |  |__/  |_
     /   |   \\____ \_/ __ \ /    \ |    |  \\__  \<   |  ||  | |  |/ ___\|  |  \   __\
    /    |    \  |_> >  ___/|   |  \|     `   \/ __ \\___  ||  |_|  / /_/  >   Y  \  |
    \_______  /   __/ \___  >___|  /_______  (____  / ____||____/__\___  /|___|  /__|
            \/|__|        \/     \/        \/     \/\/                /_____/     \/


Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.

opendaylight-user@root>
```

Lampiran 3.9 Run OpenDaylight

```
"Node: h1"
 25] 42.0-43.0 sec  16385 KBytes  134229 Kbits/sec   0.003 ms   22/11436 (0.19%

 25] 43.0-44.0 sec  15769 KBytes  129184 Kbits/sec   0.004 ms  428/11413 (3.8%)
 25] 44.0-45.0 sec  16143 KBytes  132241 Kbits/sec   0.012 ms  168/11413 (1.5%)
 25] 45.0-46.0 sec  15866 KBytes  129972 Kbits/sec   0.007 ms  362/11414 (3.2%)
 25] 46.0-47.0 sec  16092 KBytes  131830 Kbits/sec   0.007 ms  198/11408 (1.7%)
 25] 47.0-48.0 sec  16127 KBytes  132112 Kbits/sec   0.005 ms  178/11412 (1.6%)
 25] 48.0-49.0 sec  16180 KBytes  132547 Kbits/sec   0.048 ms  141/11412 (1.2%)
 25] 49.0-50.0 sec  16130 KBytes  132135 Kbits/sec   0.011 ms   58/11294 (0.51%

 25] 50.0-51.0 sec  16391 KBytes  134276 Kbits/sec   0.008 ms  111/11529 (0.96%

 25] 51.0-52.0 sec  15870 KBytes  130007 Kbits/sec   0.011 ms  260/11315 (2.3%)
 25] 52.0-53.0 sec  16181 KBytes  132559 Kbits/sec   0.008 ms  248/11520 (2.2%)
 25] 53.0-54.0 sec  16154 KBytes  132335 Kbits/sec   0.009 ms  152/11405 (1.3%)
 25] 54.0-55.0 sec  16144 KBytes  132253 Kbits/sec   0.009 ms  130/11376 (1.1%)
 25] 55.0-56.0 sec  16207 KBytes  132770 Kbits/sec   0.014 ms  169/11459 (1.5%)
 25] 56.0-57.0 sec  16028 KBytes  131300 Kbits/sec   0.007 ms  214/11379 (1.9%)
 25] 57.0-58.0 sec  16176 KBytes  132512 Kbits/sec   0.009 ms  138/11406 (1.2%)
 25] 58.0-59.0 sec  16108 KBytes  131959 Kbits/sec   0.008 ms  230/11451 (2%)
 25]  0.0-60.0 sec  964031 KBytes  131623 Kbits/sec   0.008 ms 13244/684787 (1.
%)
 25] 0.0000-59.9999 sec  483 datagrams received out-of-order
"Node: h2"
 25] 43.0-44.0 sec  16382 KBytes  134205 Kbits/sec
 25] 44.0-45.0 sec  16377 KBytes  134158 Kbits/sec
 25] 45.0-46.0 sec  16394 KBytes  134299 Kbits/sec
 25] 46.0-47.0 sec  16381 KBytes  134193 Kbits/sec
 25] 47.0-48.0 sec  16382 KBytes  134205 Kbits/sec
 25] 48.0-49.0 sec  16382 KBytes  134205 Kbits/sec
 25] 49.0-50.0 sec  16213 KBytes  132817 Kbits/sec
 25] 50.0-51.0 sec  16550 KBytes  135581 Kbits/sec
 25] 51.0-52.0 sec  16243 KBytes  133064 Kbits/sec
 25] 52.0-53.0 sec  16535 KBytes  135452 Kbits/sec
 25] 53.0-54.0 sec  16375 KBytes  134146 Kbits/sec
 25] 54.0-55.0 sec  16331 KBytes  133782 Kbits/sec
 25] 55.0-56.0 sec  16448 KBytes  134746 Kbits/sec
 25] 56.0-57.0 sec  16337 KBytes  133829 Kbits/sec
 25] 57.0-58.0 sec  16374 KBytes  134135 Kbits/sec
 25] 58.0-59.0 sec  16438 KBytes  134664 Kbits/sec
 25] 59.0-60.0 sec  16385 KBytes  134229 Kbits/sec
 25]  0.0-60.0 sec  983044 KBytes  134218 Kbits/sec
 25] Sent 684787 datagrams
 25] Server Report:
 25]  0.0-60.0 sec  964031 KBytes  131623 Kbits/sec   0.007 ms 13244/684787 (1.
%)
 25] 0.0000-59.9999 sec  483 datagrams received out-of-order
oot@naufal-VirtualBox:/home/naufal/mininet#
```

Lampiran 3.10 Pengujian *Throughput*

Lampiran 3.11 Pengujian *Latency*

KEMENTERIAN PENDIDIKAN, KEBUDAYAAN,
RISET DAN TEKNOLOGI
UNIVERSITAS HASANUDDIN
FAKULTAS TEKNIK
Poros Malino Km.6Bontomarannu(92172) Gowa, Sulawesi Selatan 92172, Sulawesi Selatan
Telp. (0411) 586015, 586262 Fax (0411) 586015
http://eng.unhas.ac.id, Email : teknik@unhas.ac.id

SURAT PENUGASAN

No. 29045/UN4.7.1/TD.06/2023

Dari : Dekan Fakultas Teknik Universitas Hasanuddin

Kepada : 1. Dr.Eng. Ir. Muhammad Niswar, ST., M.IT — Pemb. I
2. Dr.Eng. Zulkifli Tahir, ST., M.Sc. — Pemb. II

I s I : 1. Berdasarkan Surat Ketua Departemen Teknik Informatika Fakultas Teknik Nomor. 1427/UN4.7.7./TD.06/2023 tanggal 12 Desember 2023 tentang usul DOSEN PEMBIMBING MAHASISWA, maka dengan ini kami menugaskan Saudara untuk membimbing penulisan Skripsi/Tugas Akhir mahasiswa Teknik Informatika Fakultas Teknik Universitas Hasanuddin di bawah ini :

N a m a :	No. Stambuk :
Muhammad Naufal Faliq	D121 17 1503

Judul Skripsi/Tugas Akhir :

" Analisis Kinerja Routing Protokol OSPF Menggunakan Controller Ryu dan Opendaylight pada Jaringan Software Defined Network (SDN) "

2. Surat penugasan pembimbing ini mulai berlaku sejak tanggal ditetapkannya dan berakhir sampai selesainya penulisan Skripsi/Tugas Akhir mahasiswa tersebut.

3. Agar penugasan ini dilaksanakan sebaik-baiknya dengan penuh rasa tanggung jawab.

Ditetapkan di Gowa
Pada tanggal 12 Desember 2023
a.n. Dekan,
Wakil Dekan Bidang Akademik dan Kemahasiswaan
Fakultas Teknik Unhas

Dr. Amil Ahmad Ilham, ST., M.IT
NIP. 197310101998021001

Tembusan :
1. Dekan FT-UH,
2. Ketua Departemen Teknik Informatika FT-UH,
3. Mahasiswa yang bersangkutan

KARTU BIMBINGAN SKRIPSI

Prodi S1 Teknik Informatika Universitas Hasanuddin

Stb.	Nama Mahasiswa
D121171503	Muh. Naufal Faliq

Pembimbing	Nama Pembimbing	Paraf & Tgl. Persetujuan Ujian Akhir
I	Dr. Eng. Muhammad Niswar, S.T., M.IT	
II	Dr. Eng. Zulkifli Tahir, S.T., M.Sc	
No. SK Pemb		

Judul Skripsi	ANALISIS KINERJA PROTOKOL ROUTING OSPF MENGGUNAKAN CONTROLLER RYU DAN OPENDAYLIGHT PADA JARINGAN SOFTWARE DEFINED NETWORK (SDN)

No	Tanggal Bimbingan	Uraian Kegiatan Bimbingan	Paraf Pemb.
1	19 Maret 2024	Bimbingan mengenai sistem	
2	27 April 2024	Konsultasi mengenai topologi	
3	13 Mei 2024	Presentasi sistem	
4	3 Juni 2024	Presentasi mengenai OSPF pada sistem	
5			
6			
7			
8	23 Februari 2024	Konsultasi mengenai sistem	
9	15 Maret 2024	Konsultasi mengenai pengujian parameter	
10	14 Mei 2024	Konsultasi sistem	
11	4 Juni 2024	Konsul penulisan	
12	14 Juni 2024	Konsultasi penulisan	
13			
14			
15			

LEMBAR PERBAIKAN SKRIPSI

"ANALISIS KINERJA PROTOKOL ROUTING OSPF MENGGUNAKAN CONTROLLER RYU DAN OPENDAYLIGHT PADA JARINGAN SOFTWARE DEFINED NETWORK (SDN)"

OLEH:

MUHAMMAD NAUFAL FALIQ
D121171520

Skripsi ini telah dipertahankan pada Ujian Akhir Sarjana pada tanggal 31 Juli 2024. Telah dilakukan perbaikan penulisan dan isi skripsi berdasarkan usulan dari penguji dan pembimbing skripsi.

Persetujuan perbaikan oleh tim penguji:

	Nama	Tanda Tangan
Ketua	Dr-Eng.Ir. Muhammad Niswar, ST, M.InfoTech	
Sekretaris	Dr.Eng. Zulkifli Tahir, S.T., M.Sc.	
Anggota	Adnan, S.T, M.T, Ph.D	
	Iqra Aswad, S.T., M.T.	

Persetujuan perbaikan oleh pembimbing:

Pembimbing	Nama	Tanda Tangan
I	Dr-Eng.Ir. Muhammad Niswar, ST, M.InfoTech	
II	Dr.Eng. Zulkifli Tahir, S.T., M.Sc.	