TUGAS AKHIR – KI141502

IMPLEMENTASI MANAJEMEN TRANSFER RATE PADA PROSES HDFS HADOOP BERBASIS SDN

Narendra Hanif Wicaksana
NRP 5112100160

Dosen Pembimbing
Ir. F.X. Arunanto, M.Sc.
Hudan Studiawan, S.Kom., M.Kom.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2016

[Halaman ini sengaja dikosongkan]

FINAL PROJECT – KI141502

TRANSFER RATE MANAGEMENT ON HADOOP HDFS PROCESS IN A SDN-BASED IMPLEMENTATION

Narendra Hanif Wicaksana
NRP 5112100160

Advisor
Ir. F.X. Arunanto, M.Sc.
Hudan Studiawan, S.Kom., M.Kom.

DEPARTMENT OF INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2016

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

IMPLEMENTASI MANAJEMEN TRANSFER RATE PADA PROSES HDFS HADOOP BERBASIS SDN

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Komputasi Berbasis Jaringan
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh :
Narendra Hanif Wicaksana
NRP:5112100160

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Ir. F.X. Arunanto, M.Sc. .....
NIP: 195701011983031004 (pembimbing 1)

Hudan Studiawan, S.Kom., M.Kom .....
NIP: 198705112012121003 (pembimbing 2)

SURABAYA
JUNI 2016

[Halaman ini sengaja dikosongkan]

IMPLEMENTASI MANAJEMEN TRANSFER RATE PADA PROSES HADOOP HDFS BERBASIS SDN

Nama Mahasiswa : Narendra Hanif
NRP : 5112100160
Jurusan : Teknik Informatika FTIf-ITS
Dosen Pembimbing 1 : Ir. F.X. Arunanto, M.Sc.
Dosen Pembimbing 2 : Hudan Studiawan, S.Kom., M.Kom.

ABSTRAK

Hadoop dan Software Defined Networking merupakan dua teknologi yang sedang berkembang saat ini. Hadoop sebagai platform pengolahan big data dan Software Defined Networking arsitektur jaringan. Switch OpenFlow merupakan salah satu komponen utama pada arsitektur jaringan Software Defined Networking. Telah banyak dilakukan penelitian memanfaatkan Software Defined Networking dalam memanajemen jaringan, salah satunya adalah optimasi proses MapReduce pada klaster Hadoop. Infrastruktur arsitektur SDN juga merupakan salah satu permasalahan dikarenakan memerlukan switch yang mendukung OpenFlow. Saat ini terdapat metode untuk membuat switch OpenFlow dengan menggunakan komputer berbasis Linux.

Pada Tugas Akhir akan dilakukan implementasi manajemen transfer rate pada proses penyimpanan data HDFS Hadoop. Manajemen transfer dilakukan dengan cara melakukan kategori lalu lintas jaringan pada klaster Hadoop dengan memanfaatkan fitur queue yang dimiliki oleh switch OpenFlow. Selain itu juga dibangun arsitektur Softwared Defined Networking menggunakan Raspberry Pi sebagai switch OpenFlow.

Dari hasil dari implementasi ini didapatkan perbedaan waktu pada proses HDFS Hadoop antara dilakukan pengaturan

manajemen transfer rate dan tanpa dilakukan manajemen transfer rate. Waktu Proses HDFS ketika terdapat lalu lintas lain atau congestion pada klaster Hadoop mengalami penurunan bila dilakukan manajemen transfer rate.

Kata kunci**: **Software Defined Networking, OpenFlow, Hadoop, HDFS

TRANSFER RATE MANAGEMENT ON HADOOP HDFS PROCESS IN A SDN-BASED IMPLEMENTATION

Student Name : Narendra Hanif Wicaksana
Student ID : 5112100160
Major : Teknik Informatika FTIf-ITS
Advisor 1 : Ir. F.X. Arunanto, M.Sc.
Advisor 2 : Hudan Studiawan, S.Kom., M.Kom.

ABSTRACT

Hadoop and Software Defined Networking are emerging technologies at this time. Hadoop as a platform for big data processing and Software Defined Networking new network architecture. OpenFlow switch is one of the major components on the Software Defined Networking. A lot of research utilizing of the Software Defined Networking in management of the network, one of which is the optimization of the process of MapReduce in Hadoop clusters. SDN architecture infrastructure is also one of the problems due to the need to support OpenFlow switches. Currently there are methods to make OpenFlow switch to using Linux-based computers.

In this the final project will be the implementing transfer rate management on Hadoop HDFS process. Management transfer is performed by categorized network traffic on a Hadoop cluster with queue. In this final project will build SDN infrastructure using Raspberry Pi as OpenFlow switches.

The results of this implementation show decrease time on Hadoop HDFS with tranfer rate management.Data transfer for HDFS get more transfer rate than other traffic.

Keywords**: **Software Defined Networking, OpenFlow, Hadoop

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT atas segala karunia dan rahmat-Nya penulis dapat menyelesaikan Tugas Akhir yang berjudul :

"Implementasi Manajemen *Transfer Rate* pada Proses HDFS Hadoop Berbasis SDN"

Harapan dari penulis semoga apa yang tertulis di dalam buku Tugas Akhir ini dapat bermanfaat bagi pengembangan ilmu pengetahuan saat ini, serta dapat memberikan kontribusi yang nyata.

Dalam pelaksanaan dan pembuatan Tugas Akhir ini tentunya sangat banyak bantuan yang penulis terima dari berbagai pihak, tanpa mengurangi rasa hormat penulis ingin mengucapkan terima kasih sebesar-besarnya kepada:

1. Allah SWT dan Nabi Muhammad SAW.
2. Papa, Mama, kedua kakak penulis, Nabila dan Ninda, dan juga Fatio yang selalu memberikan dukungan, baik moral maupun material, secara penuh untuk menyelesaikan Tugas Akhir ini.
3. Bapak Ir. F.X. Arunanto, M.Sc. dan Bapak Hudan Studiawan, S.Kom., M.Kom. selaku dosen pembimbing yang selalu membantu, menyemangati dan memotivasi penulis dengan ilmu-ilmu dalam pengerjaan Tugas Akhir.
4. Bapak/Ibu dosen, staf dan karyawan Jurusan Teknik Informatika ITS yang telah banyak memberikan dukungan, ilmu dan bimbingan yang tak ternilai harganya bagi penulis.
5. Teman-teman di Kos Bunda yang dapat memberikan pencerahan-pencerahan ketika penulis sedang kesulitan dalam mengerjakan Tugas Akhir ini.

6. Serta pihak-pihak lain yang tidak dapat disebutkan disini yang telah banyak membantu penulis dalam penyusunan Tugas Akhir ini.

Penulis telah berusaha sebaik-baiknya dalam menyusun Tugas Akhir ini, namun penulis mohon maaf apabila terdapat kekurangan, kesalahan maupun kelalaian yang telah penulis lakukan. Kritik dan saran yang membangun dapat disampaikan sebagai bahan perbaikan selanjutnya.

Surabaya, Juni 2016
Penulis

Narendra Hanif Wicaksana

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

Pada bab ini dibahas mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika laporan tugas akhir. Diharapkan dari penjelasan dalam bab ini gambaran Tugas Akhir secara umum dapat dipahami.

1.1. Latar Belakang

Saat ini berkembang arsitektur baru dalam mengelola jaringan yang dikenal sebagai *Software Defined Networking* [1]. Dimana kontrol jaringan yang sebelumnya berada pada tiap perangkat, dipindahkan menjadi terpusat. Ide *Software Defined Networking* muncul dari prinsip jaringan yang dapat diprogram (*programmable network*) sehingga konfigurasi perangkat jaringan menjadi lebih fleksibel, dinamis dan dapat diprogram. Hal tersebut dilakukan dengan cara mengambil alih proses penanganan paket (*packet handling*) dari yang sebelumnya dilakukan ditiap perangkat menjadi terpusat. Aristektur *Software Defined Networking* dibagi menjadi tiga komponen, yaitu infrastruktur, kontrol dan juga aplikasi. Konsep pada *Software Defined Networking* dapat mempermudah dan mempercepat inovasi pada jaringan.

Salah satu hal yang terpenting dalam implementasi *Software Defined Networking* adalah OpenFlow [2]. OpenFlow merupakan salah satu antarmuka antara komponen kontroler dan infrastruktur, atau yang dikenal dengan antarmuka *Southbound*. Tidak semua perangkat jaringan mendukung OpenFlow. Oleh karena itu dibutuhkan switch OpenFlow untuk mengimplementasikan *Software Defined Networking*.

Pada switch atau router saat ini *data path* dan juga *control path* berada pada perangkat yang sama. Switch OpenFlow memisahkan dua fungsi tersebut. *Data path* masih tetap berada pada switch sedangkan keputusan pemilihan rute dipindahkan ke kontroler terpisah. Ketika switch OpenFlow menerima paket yang

baru diterima yang tidak berada pada alur masuk (*flow entry*), paket akan dikirimkan ke kontroler. Kontroler akan mengambil keputusan bagaimana menangani paket tersebut. Paket dapat di *drop* atau dapat juga dimasukan kedalam alur masuk switch, sehingga switch dapat mengetahui bagaimana jika mendapatkan paket yang serupa.

Fitur lain yang dimiliki oleh OpenFlow adalah pengaturan *transfer rate*. Switch OpenFlow dapat melakukan pengaturan *transfer rate* dengan cara pembuatan *queue*. *Queue* yang dibuat dapat diatur nilai maksimum dan juga nilai minimum *transfer rate*. Satu atau lebih *queue* dapat diberikan tiap port yang nantinya diterapkan pada pemetaan alur port tersebut.

Hadoop merupakan *platform* pengolaahan big data yang sedang berkembang saat ini. Hadoop menyediakan mekanisme penyimpanan data dan juga pengolahan data. Data-data pada klaster Hadoop disimpan pada sistem file yang dikenal dengan sebutan HDFS (*Hadoop Distributed File System*). Pengolah data pada Hadoop menggunakan model pemograman MapReduce.

Pada kluster Hadoop perpindahan data akan sering terjadi, karena data yang disimpan akan tersebar ke dalam Datanode terutama pada saat melakukan proses penyimpanan pada ke dalam HDFS. Dengan melakukan pengaturan *transfer rate* diharapkan dapat meningkatkan performa HDFS Hadoop.

Memanfaatkan fitur yang dimiliki oleh OpenFlow, pada Tugas Akhir ini mencoba mengoptimalkan proses HDFS dengan cara melakukan manajemen *transfer rate*. Mengimplementasikan pengaturan *transfer rate* dengan cara mengatur *queue* pada klaster Hadoop. Penelitiaan mengenai optimasi klaster Hadoop dengan memanfaatkan OpenFlow pernah dilakukan oleh S. Narayan [2]. Penelitian dilakukan dengan cara melakukan pengaturan *queue* pada proses MapReduce Hadoop.

Untuk dapat mengimplementasikan *Software Defined Networking* dibutuhkan switch OpenFlow. Pada Tugas Akhir ini juga akan dibahas bagaimana membangun infrastruktur SDN dengan menggunakan *low-cost embedded* Linux *machine* sebagai

switch OpenFlow. Sehingga dapat mendukung pengerjaan Tugas Akhir ini tanpa perlu mengubah arsitektur jaringan sekarang menjadi *Software Defined Networking*, dimana untuk mengubah arsitektur jaringan menjadi *Software Defined Networking* terdapat biaya pergantian infrastruktur yang tidak sedikit. Penggunaan *low-cost embedded* Linux *machine* sebagai switch *Software Defined Networking* pernah juga dilakukan oleh K. Hyunmin [4] yang menggunakan Raspberry Pi sebagai switch OpenFlow

Pada Tugas Akhir ini akan mengimplementasikan manajemen *transfer rate* pada proses HDFS Hadoop berbasis OpenFlow. Adapun switch OpenFlow yang digunakan adalah Raspberry Pi. Dengan adanya Tugas Akhir ini diharapkan dapat menjadi salah satu alternatif pemanfaatan arsitektur jaringan *Software Defined Networking* pada klaster Hadoop.

1.2. Rumusan Permasalahan

Berdasarkan uraian di atas dapat dirumuskan hal-hal sebagai berikut:

1. Bagaimana membangun arsitektur *Software Defined Networking* ?
2. Bagaimana cara mengimplementasikan fitur pengaturan *transfer rate* pada switch OpenFlow ?
3. Bagaimana memanfaatkan pengaturan *transfer rate* pada proses HDFS Hadoop?

1.3. Batasan Permasalahan

Beberapa batasan masalah yang terdapat dalam tugas akhir ini dapat dipaparkan sebagai berikut:

1. Infrastruktur yang dibangun menggunakan perangkat keras Raspberry Pi.
2. Menggunakan perangkat lunak Open vSwitch untuk setiap perangkat switch dan menggunkan OpenFlow versi 1.3.
3. Manajemen *transfer rate* dilakukan dengan mengatur *queue* pada switch OpenFlow.

4. Penambahan lalu lintas jaringan menggunakan Iperf.
5. Menggunakan kontroler *Software Defined Networking* Ryu, yang menggunkan Bahasa pemograman Python
6. Menggunakan Hadoop versi 1.2.

1.4. Tujuan

Tujuan dari pembuatan tugas akhir ini antara lain :

1. Mengoptimalkan proses HDFS Hadoop pada klaster Hadoop berbasis OpenFlow.
2. Membangun infrastruktur *Software Defined Networking* menggunkan *low-cost embedded* Linux *machine.*

1.5. Manfaat

Tujuan dari pembuatan tugas akhir ini antara lain :

1. Solusi pemanfaatan switch OpenFlow untuk klaster Hadoop.
2. Memberikan solusi alternatif untuk membangun infrastruktur SDN sebagai sarana riset dan juga pembelajaran tanpa perlu mengubah infrastruktur yang ada menjadi *Software Defined Networking*.

1.6. Metodologi

Pembuatan Tugas Akhir dilakukan menggunakan metodologi sebagai berikut

1.6.1. Penyusunan proposal Tugas Akhir

Proposal tugas akhir ini berisi tentang deskripsi pendahuluan dari tugas akhir yang akan dibuat. Pendahuluan pada proposal tugas akhir ini terdiri dari latar belakang diajukannya usulan tugas akhir, rumusan masalah yang diangkat, batasan masalah untuk tugas akhir, tujuan dari pembuatan tugas akhir, dan manfaat hasil dari pembuatan tugas akhir. Selain itu dijelaskan pula tinjauan pustaka yang digunakan sebagai referensi pendukung implementasi tugas

akhir. Pada proposal ini juga terdapat perencanaan jadwal pengerjaan tugas akhir.

1.6.2. Studi literatur

Pada tahap ini dilakukan pencarian, pengumpulan, penyaringan, pemahaman, dan pembelajaran literatur yang berhubungan dengan *Software Defined Networking*, OpenFlow, Hadoop. Literatur yang digunakan meliputi: buku referensi, jurnal, dan dokumentasi internet.

1.6.3. Perancangan

Adapun fitur yang terdapat dalam aplikasi ini nantinya antara lain:

1. Perancangan Arsitektur *Software Defined Networking*
2. Perancangan Arsitektur Hadoop
3. Perancangan fitur pengaturan *transfer rate* pada klaster Hadoop.

1.6.4. Implementasi

Pada tahap ini dilakukan implementasi proses yang telah didefinisikan pada bab Perancangan. Tahapan pertama adalah mengimplementasikan arsitektur *Software Defined Networking*, kemudian implementasi klaster Hadoop dan fitur pengaturan *transfer rate* pada klaster Hadoop.

1.6.5. Pengujian dan Evaluasi

Pada tahapan ini dilakukan pengujian terhadap sistem yang dibangun. Pengujian manajeman *transfer rate* dilakukan dengan cara memasukan lalu lintas jaringan tambahan pada kalster Hadoop.Hasil pengujian aplikasi akan diberikan evaluasi serta ditarik kesimpulan untuk kemudian diujikan kembali untuk memastikan aplikasi telah berfungsi sebagaimana mestinya

1.6.6. Penyusunan Buku Tugas Akhir

Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini serta hasil dari implementasi aplikasi perangkat lunak yang telah dibuat.

1.7. Sistematika Penulisan

Buku Tugas akhir ini bertujuan untuk memberikan gambaran dari pengerjaan tugas akhir ini. Selain itu dimana dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku tugas akhir ini terdiri dari beberapa bagian seperti berikut:

Bab I Pendahuluan

Bab ini berisi latar belakang masalah, rumusan dan batasan permasalahan, tujuan dan manfaat pembuatan Tugas Akhir, metodologi yang digunakan, dan sistematika penyusunan Tugas Akhir.

Bab II Tinjauan Pustaka

Bab ini membahas dasar pembuatan dan beberapa teori penunjang yang berhubungan dengan pokok pembahasan yang mendasari pembuatan Tugas Akhir ini.

Bab III Perancangan

Bab ini membahas tentang deskripsi umum sistem yang digunakan dan rancangan dari perangkat platform yang akan dibangun sesuai dengan yang telah teori teori yang telah dijelaskan pada bab tinjauan pustaka. Rancangan meliputi rancangan secara logik dan juga rancangan fisik.

Bab IV Implementasi

Bab ini membahas tentang implementasi dari perancangan pada bab sebelumnya. Bagaimana mengimplementasikan SDN pada perangkat Raspberry Pi, seperti konfigurasi apa saja yang harus dilakuakn instalasi tools yang aakan digunakan.

Bab V Uji Coba dan Evaluasi

Bab ini membahas tentang pengujian dari platform SDN yang dibuat dan metode yang dikembangkan. Melakukan evaluasi platform yang dibangun apakah telah memenuhi kebutuhan serta apakah telah memenuhi harapan.

Bab VI Kesimpulan dan Saran

Bab ini berisi kesimpulan dari hasil pengujian yang dilakukan serta saran untuk pengembangan aplikasi selanjutnya.

Daftar Pustaka

Merupakan daftar refrensi yang digunakan dalam pembuatan tugas akhir.

Lampiran

Merupakan bab tambahan yang berisi hasil uji coba yang telah dilakukan.

[Halaman ini sengaja dikosongkan]

BAB II
TINJAUAN PUSTAKA

Bab ini menjelaskan tentang tinjauan pustaka yang menjadi dasar pembuatan Tugas Akhir. Beberapa teori, pustaka, dan teknologi yang mendasari pengerjaan Tugas Akhir ini. Penjelasan secara khusus masing-masing tinjauan pustaka dapat dilihat pada masing-masing subbab berikut ini.

2.1. *Software Defined Networking*

Software Defined Networking (SDN) [1] adalah teknologi yang relatif baru, muncul dari prinsip jaringan yang dapat diprogram (*programmable network*). Ide SDN muncul untuk mencoba memenuhi kebutuhan konfigurasi perangkat jaringan sehingga menjadi lebih fleksibel dan juga dinamis. Hal tersebut dilakukan dengan cara mengambil alih proses penanganan paket (*packet handling*) dari yang sebelumnya dilakukan ditiap perangkat menjadi terpusat.

Pada [5] dijelaskan pengertian SDN, yaitu pemisihan fisik *control plane* jaringan dari *forwarding plane*, dimana *control plane* dapat mengatur beberapa perangkat. Dipaparkan juga arsitektur SDN memiliki beberapa sifat:

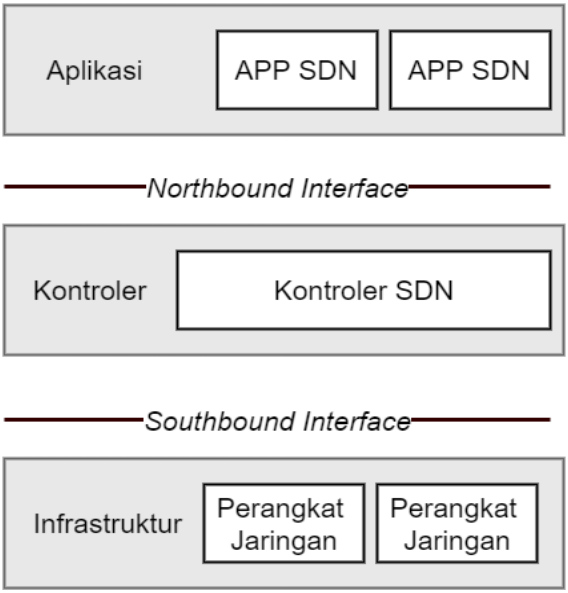
1. Dapat langsung diprogram, kontrol jaringan secara langsung dapat diprogram dikarenakan dipisahkan dengan fungsi *forwarding*.
2. *Agile*, administrator dapat menyesuaikan secara dinamis arus lalu lintas jaringan untuk memenuhi kebutuhan perubahan.
3. Dikelola secara terpusat, logika jaringan terpusat di kontroler SDN, dapat mengelola jaringan secara keseluruhan
4. Dapat dikonfigurasikan melalui program, SDN memungkinkan pengatur jaringan untuk mengkonfigurasi, mengatur, mengamankan, dan

mengoptimalkan sumber daya jaringan dengan cepat melalui program SDN, yang dapat ditulis sendiri.

5. *Open standard-based* dan tidak terikat pada vendor, SDN mensimplifikasi desain dan juga operasi jaringan,dikarenakan instruksi atau perintah diberikan oleh kontroler SDN, bukan dari perangkat dan juga protokol merk vendor khusus. Sehingga ketika mengimplementasikan SDN, tidak terikat pada satu vendor tertentu.

Tujuan dari SDN adalah menyediakan antar muka terbuka (*open interfaces*) yang dapat digunakan untuk mengembangkan perangkat lunak, untuk dapat mengatur konektivitas dan juga alur lalu lintas jaringan pada perangkat jaringan yang ada.

Gambar 2.1 menampilkan komponen dasar dari SDN. Terdapat tiga komponen dasar dalam SDN, dimana diantara tiap komponen terdapat antarmuka.



Gambar 2.1 Komponen SDN

Komponen pertama adalah aplikasi (*application layer/ application plane*). Aplikasi berada pada lapisan paling atas, aplikasi-aplikasi SDN berada pada bagian aplikasi, beberapa aplikasi dapat saling berjalan dan berkolaborasi satu sama lain. Aplikasi-aplikasi ini berkomunikasi dengan kontroler menggunakan API, API ini sering disebut sebagai *Northbound interface* atau juga sering disebut dengan NBI.

Komponen kedua adalah kontroler (*control layer/ controller plane*). Kontroler SDN akan menerjemahkan kebutuhan aplikasi dengan infrastruktur dengan memberikan instruksi yang sesuai untuk SDN *datapath* serta memberikan informasi yang relevan dan dibutuhkan oleh lapisan aplikasi.

Komponen ketiga adalah infrastruktur (*infastructure layer/ data plane*). Terdiri dari elemen-elemen jaringan yang dapat mengatur SDN *datapath* sesuai dengan instruksi yang diberikan oleh kontroler. *Control plane* mengirim intruksi ke *infrastructure layer* melalui antarmuka yang disebut dengan *Soutbound interface.*

Dalam [1] dijelaskan *use case* penerapan SDN untuk berbagai macam segmen seperti *data center, cloud,* perusahaan telekomunikasi hingga universitas.

Perkembangan virtualisasi server, *IT- as-a Service,* penggunaan perangkat *mobile,* dan perubahan perubahan dengan cepat untuk merespon kebutuhan bisnis merupakan tuntutan-tuntutan jaringan pada saat ini. *Software Defined Networking* muncul dengan menyediakan arsitektur jaringan dinamis yang mengubah jaringan menjadi platform *rich service delivery.*

Dengan pemisahan kontrol jaringan dan juga *data plane,* arsitektur SDN berbasis OpenFlow dapat diterapkan baik di lingkungan *enterprise,* pusat data (*data center*) dan juga universitas. Mengadopsi SDN dapat meningkatkan pengelolaan jaringan menjadi *manageability, scalability,* dan juga *agile.* Pemanfaatan SDN banyak keuntungan dan juga momentum perkembangan industri saat ini, SDN menjadi *new form for network*

2.2. OpenFlow

Ide dari OpenFlow [2] muncul dari perkembangan pada saat ini dimana switch dan juga router memiliki *flow-table*. Namun tiap vendor memiliki *flow-table* yang berbeda. OpenFlow mencoba menyelesaikan permasalahan tersebut dengan cara menyediakan *open-protocol* untuk memprogram switch dan juga router yang berbeda dan tidak terikat pada vendor.

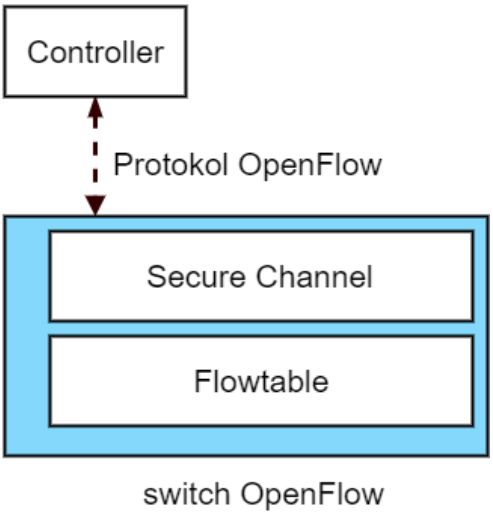
OpenFlow merupakan salah satu komponen dari arsitektur SDN. OpenFlow dikembangkan pada tahun 2009 oleh universitas Stanford. OpenFlow merupakan hal yang terpenting dalam implementasi SDN. OpenFlow merupakan protokol komunikasi atau yang disebut *Southbound interface*, antara *control plane* dengan *forwarding plane* sehingga memungkinkan untuk memprogram *flow table* pada switch dan router. Menggunakan OpenFlow dapat langsung memanipulasi *forwarding plane* pada perangkat jaringan seperti switch dan router.

Dengan menggunakan OpenFlow memungkinkan para peneliti untuk mengkustomasi alur (*flow)*, dengan cara memlih rute untuk paket dan memproses penerimaan. Sehingga memungkinkan peneliti untuk membuat *routing protocol* baru, model *security*, dan juga skema pemberian alamat (*addressing schemes*).

Komponen OpenFlow seperti yang dapat kita lihat pada Gambar 2.2 terdiri dari tiga komponen:

1. *Flow Table*,
2. *Secure Channel*
3. Protokol OpenFlow.

Tiga komponen OpenFlow: (1) *Flow Table* berhubungan dengan pengaturan alur yang masuk, untuk memberi tahu kepada switch bagaimana untuk memproses alur. (2) *Secure Channel* yang menghubungkan antara switch dengan kontroler untuk pengiriman paket dan juga perintah menggunakan (3) protokol OpenFlow, yang menyediakan tata cara untuk berkomunikasi antara switch dan juga kontroler.



Gambar 2.2 Komponen OpenFlow

Arsitektur SDN menggunkan protokol OpenFlow untuk berkomunikasi antara kontroler dengan switch OpenFlow. Protokol ini dapat mengubah *flow table* dari switch OpenFlow. Ketika switch OpenFlow menerima paket, switch akan mencoba mencocokan nilai *header* dari paket dengan aturan yang telah ada pada *flow table*. Apabila paket tidak ditemukan pada *flow table* maka switch OpenFlow, akan mengirimkan ke kontroler. Kontroler akan mengambil keputusan bagaimana menangani paket tersebut. Paket dapat di *drop* atau dapat juga dimasukan kedalam alur masuk switch, sehingga berikutnya switch dapat mengetahui bagaimana jika mendapatkan paket yang serupa.

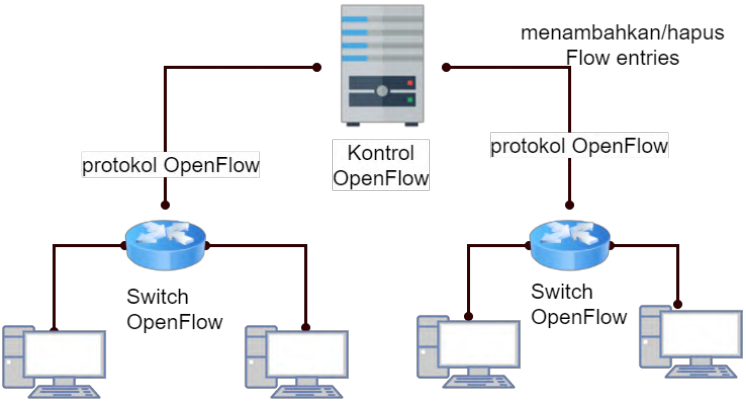
Flow table seperti yang dapat dilihat pada Gambar 2.3 berisi kumpulan aturan yang menjelaskan bagaimana paket akan diproses ketika diterima. Aturan pada *flow table* dapat ditambah, modifikasi, dan juga dihapus oleh kontroler yang disebut sebagai *flow entries*. *Flow entries* terdiri dari tiga bagian, *Match fields* merepresentasikan paket *header* yang menjelaskan flow. Action berisi perintah yang dilakukan bila paket terdapat pada *match field*,

action dapat berisi perintah untuk *drop* paket, modifikasi paket *header*. Statistik yang menghitung jumlah paket dan juga *bytes* pada tiap *flow*.

Gambar 2.3 Flow Table Header OpenFlow

Kontroler juga merupakan bagian dalam arsitektur OpenFlow seperti yang dapat kita lihat pada Gambar 2.4. Kontroler dapat terhubung dengan satu atau lebih switch melalui *channel* dan juga protokol OpenFlow. Kontroler dapat menerima permintaan bagaimana menangani paket dan memberikan instruksi kepada switch untuk bagaimana memproses paket tersebut. Pesan pada OpenFlow protokol terdiri dari tiga jenis:

1. Kontroler ke switch
2. *Asynchrounous*
3. *Symetric*



Gambar 2.4 Gambaran Kontroler OpenFlow

Pesan yang dikirim dari kontroler ke switch:

1. Menjelaskan, memodifikasi, menghapus *flow*.
2. Meminta informasi mengenai kapabilitas switch.
3. Menerima informasi *counter* dari switch.
4. Mengirim paket ke switch untuk pemprosesan flow baru.

Pesan *asynchrounous* dikirimkan oleh switch untuk:

1. Mengirim kepada kontroler paket yang tidak terdapat pada *flow table*.
2. Memberikan informasi kepada kontroler bahwa *flow* telah dihapus karena parameter *time to live* telah habis.
3. Memberikan informasi ke kontroler mengenai perubahan status *port*.

Pesan symmetric dapat dikirim baik oleh switch maupun kontroler dan digunakan untuk:

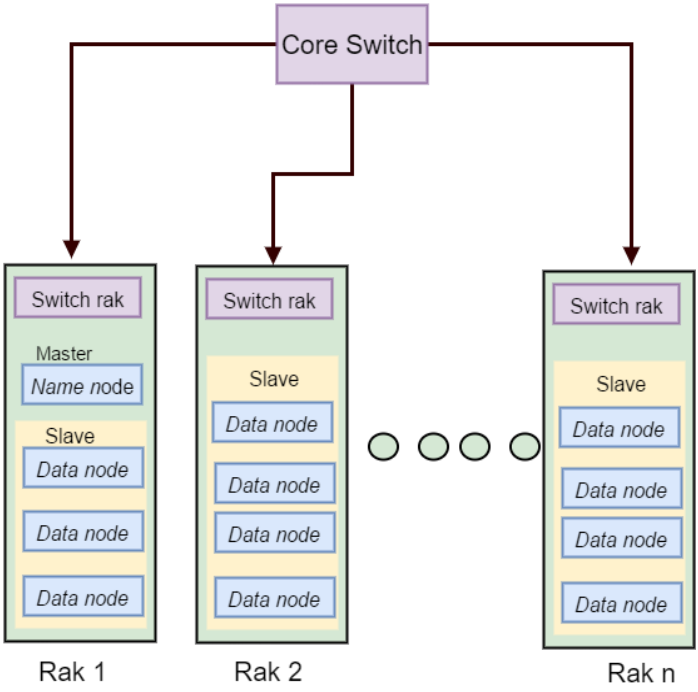
1. Pertukaran *hello message* antar switch dan juga kontroler pada awal.
2. *Echo* untuk mengetes apakah koneksi antar switch dan juga kontroler masih tersambung

Berdasarkan hal-tersebut OpenFlow dapat mempermudah inovasi dalam jaringan. Fitur lain yang dimiliki oleh OpenFlow adalah pengaturan *quality-of-service* (QoS). Switch OpenFlow dapat melakukan pengaturan QoS dengan cara pembuatan *queue*. *Queue* yang dibuat dapat diatur nilai maksimum dan juga nilai minimum *transfer rate*. Satu atau lebih *queue* dapat diberikan tiap port yang nantinya diterapkan pada pemetaan alur *port* tersebut. Alur yang dipetakan dengan *queue* akan diberikan sesuai dengan pengaturan QoS yang telah dibuat.

Untuk mendukung perkembangan OpenFlow dibangun konsorsium industri yang bernama Open Networking Foundation yang berisi vendor vendor perangkat jaringan terkemuka seperti Cisco, ZTE, Huawei, Juniper dan juga beberapa perusahaan lain seperti Google, Facebook, HP, Microsoft AT&T. Para anggota Open Networking Foundation telah mulai mengimplementasikan protokol OpenFlow pada infrastruktur jaringan.

2.3. Hadoop

Hadoop [6] adalah sebuah framework pengolahan *big data open source* yang dibangun dengan bahasa Java dibawah lisensi Apache. Hadoop dibuat oleh Doug Cutting, sering digunakan untuk pustaka pencarian kata. Hadoop salah satu *platform* pengolahan *big data* yang cukup dikenal pada saat ini. Hadoop menyediakan penyimpanan data dan juga penganalisaan data yang dapat dilakukan komputer secara paralel. Tiap komputer memiliki peran seperti yang dapat dilihat pada Gambar 2.5. Hadoop dapat dibangun pada perangkat keras yang umum. Hadoop berbeda dengan *database*, Hadoop lebih ke sebuah perangkat lunak yang secara khusus mengangani data dengan jumlah yang besar dan data



Gambar 2.5 Arsitektur Klaster Hadoop

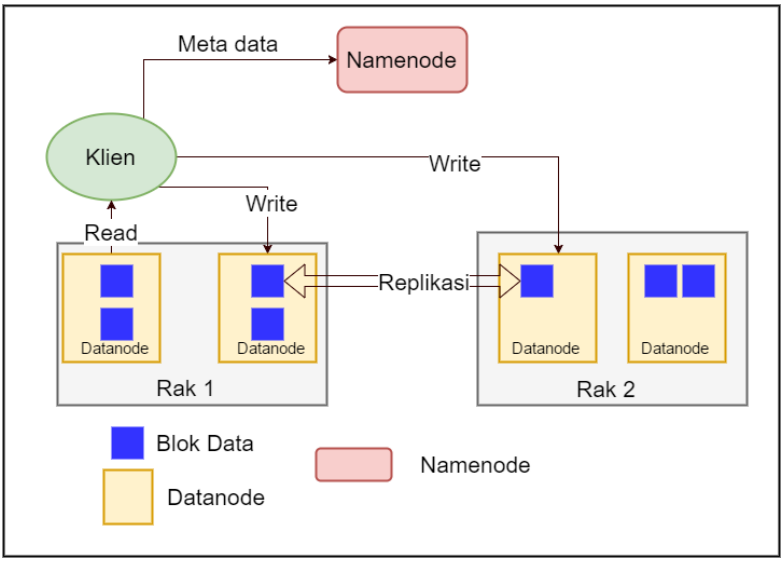
yang tidak terstruktur berbeda dengan *database* tradisional yang menangani data yang bersifat terstruktur.

Kelebihan Hadoop adalah dapat mengolah data tidak terstruktur. Berbeda dengan data terstruktur, data tidak terstruktur tidak memiliki batas-batas yang tetap pada suatu file. Data tidak terstruktur tidak dapat dimasukkan, disimpan, di-*query*-kan, dan dianalisa dengan menggunakan *database* tradisional. Data tidak terstruktur merupakan data yang datang dari berbagai macam sumber, seperti email, dokumen, teks, video, foto, audio, file, dan bahkan posting-posting dari media sosial. Hadoop memiliki kemampuan menganalisa berbagai macam data tanpa harus mengetahui strukturnya.

Hadoop dirancang untuk meningkatkan atau menggabungkan suatu *server* tunggal menjadi ribuan mesin-mesin, dengan mesin-mesin lain saling membagi kemampuan komputasi dan penyimpanannya, seperti yang dilihat pada Gambar 2.6. Dalam memproses suatu operasi atau pekerjaan Hadoop akan membagi/memecah pekerjaan menjadi proses yang lebih kecil disebut sebagai *task*, saat memproses data yang dilakukan oleh *master*, pembagian *task* dilakukan secara terdistribusi dengan memanfaatkan komputer-komputer lain yang terhubung membentuk suatu klaster yang disebut dengan *slave*.

Gambar 2.6 Klaster Hadoop

HDFS (Hadoop *Distributed File System*) adalah file sistem distribusi untuk menyimpan dan juga mengolah data pada satu klaster Hadoop. Dalam mekanisme penyimpanan HDFS yang diilustrasikan pada Gambar 2.7 terdapat satu node pusat yang disebut Namenode yang menyimpan metadata dari sistem file dan juga *node* yang lainnya yang disebut sebagai Datanode. File dalam HDFS dipecah menjadi blok-blok data yang lebih kecil, biasanya bernilai 64 MB tiap blok dan dapat juga diatur sesuai dengan keinginan. Tiap blok data akan disimpan secara terpisah pada Datanode dan akan direplikasi sesuai dengan pengaturan. Klien HDFS menghubungi Namenode untuk mendapatkan informasi lokasi tiap blok data.

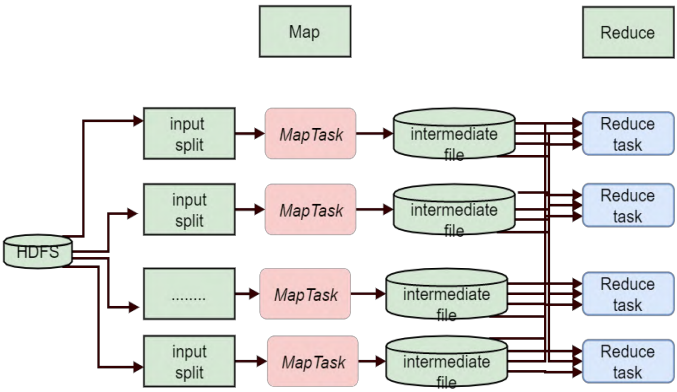


Gambar 2.7 Arsitektur HDFS

MapReduce adalah model pemograman. MapReduce terdiri dari dua fungsi yaitu *map* dan *reduce*. Pada Proses pemetaan (*mapping*), input data diambil dan diproses untuk menghasilkan *key*/*value*. Proses *reduce* mengambil *key*/*value dari* proses dari hasil beberapa pemetaan dan mengulahnya menjadi hasil akhir.

Kode program MapReduce dapat ditulis dengan berbagai macam bahasa pemograman. Program MapReduce merupakan pemograman dapat dijalankan pada komputasi paralel. Proses *map* dapat berjalan secara bersamaan dimana hasil dari *mapping* nantinya digunkan oleh proses *reduce* yang juga dapat dijalankan secara bersamaan. Terdapat dua komponen yang mengatur eksekusi pengerjaan proses MapReduce, yaitu *task tracker* dan juga *job tracker. Job tracker* merupakan komponen sentral yang mengatur proses eksekusi sementara beberapa pengaturan proses yang tersebar disebut *task tracker*.

Pada proses MapReduce terdapat dua fase pengerjaan, pada dua fase tersebut dapat terjadi perpindahan data. Pada tahap awal data input, yang disimpan dalam bentuk HDFS, dibagi menjadi beberapa split. Tiap split proses map diberikan ke *task tracker* pada Datanode terdekat. *Task tracker* akan memproses data inputan berdasarkan fungsi map yang telah dibuat, menhasilkan data dalam bentuk tupel yang berisi <*key*, *value*>. Pada fase *reduce task tracker* juga bertugas untuk melakukan pengerjaan *reduce* dengan cara mengambil data hasil proses *mapping task tracker*, menggabungkan split data hasil map, melakukn komputasi *reduce*, mengembalikan kembali data kedalam HDFS.



Gambar 2.8 Proses MapReduce

2.4. Hadoop dan SDN

Hadoop bekerja secara terdistribusi, data-data nya tersebar pada Datanode sehingga perpindahan data pada Hadoop sering terjadi. Dalam [3] dipapaparkan lalu lintas jaringan pada Hadoop dikategorikan menjadi:

1. Proses *read* dan *write* yang dilakukan oleh klien
2. Replikasi HDFS
3. Interaksi antar *task tracker*
4. Lalu lintas antara HDFS dan *tasktracker*
5. Interaksi antar Namenode dan Datanode
6. Interaksi antar *job tracker* dan juga *task tracker.*

Memanfaatakan pengkategorian jaringan yang digunakan Hadoop dapat dilakukan pengaturan lalu lintas jaringan sehingga dapat dikategorikan sesuai dengan karakteristik klaster Hadoop. Switch OpenFlow dapat melakukan pengkategorian lalu lintas jaringan sehingga dapat diatur sesuai dengan karaketeristik jaringan yang dibutuhkan.

Dalam [3] dijelaskan bagaimana memanfaatkan fitur pengaturan *transfer rate* yang dimiliki oleh OpenFlow untuk dapat memaksimalkan klaster Hadoop yang dibangun dengan cara mengkategorikan lalu lintas jaringan dengan membuat *queue.*

2.5. Perangkat Lunak Switch OpenFlow

Salah satu permasalahan dalam implementasi OpenFlow adalah memerlukan switch OpenFlow. Salah satu alternatif selain membeli perangkat switch adalah menggunkan *firmware* OpenFlow. Beberapa perangkat dapat diubah menjadi perangkat yang mendukung OpenFlow.

2.5.1. Stanford Reference Switch

Standford Reference Switch merupakan salah satu implementasi paling awal dari OpenFlow. Stanford Reference Switch dikenalakan bersamaan dengan pengenalan OpenFlow

pada tahun 2009. Perangkat lunak ini mendukung OpenFlow versi 1.0 kemudian mengalami perbaruan sehingga juga mendukung OpenFlow versi 1.1. Perangkat lunak ini mulai jarang digunakan karena tidak dikelola lagi sejak tahun 2011.

2.5.2. Pantou

Pantou [7] merupakan adapatasi dari Stanford Reference Switch untuk OpenWRT. Pantou mendukung OpenFlow versi 1.0. Pantou juga kurang direkomendasikan karena sudah tidak dikembangkan lagi sejak 2011.

2.5.3. Open vSwitch

Open vSwitch [8] atau yang dikenal dengan istilah OVS adalah implementasi virtualisasi multilayer switch, sehingga memungkinkan untuk membuat switch OpenFlow secara virtual. Tujuan utama dari Open vSwitch menyediakan switch untuk lingkungan virtualisasi perangkat keras. Dengan memanfaatkan Open vSwitch dapat membuat komputer berbasis Linux menjadi switch.

Terdapat beberapa fitur yang dapat dilakukan oleh Open vSwitch antara lain: (1) *tunneling* (seperti GRE, VXLAN, IPSEC), (2) pengaturan QoS, (3) LACP, (4) *flow export,* dan juga (4) *port mirroring*. Open vSwitch dapat diunduh pada distro Linux.

Tabel 2.1 Open vSwitch dan Versi OpenFlow

Open vSwitch	Versi OpenFlow					
	1.0	1.1	1.2	1.3	1.4	1.5
2.0	✓	●	●	●	-	-
2.1	✓	●	●	●	-	-
2.2	✓	●	●	●	●	●
2.3	✓	✓	✓	✓	●	●

✓ = Didukung

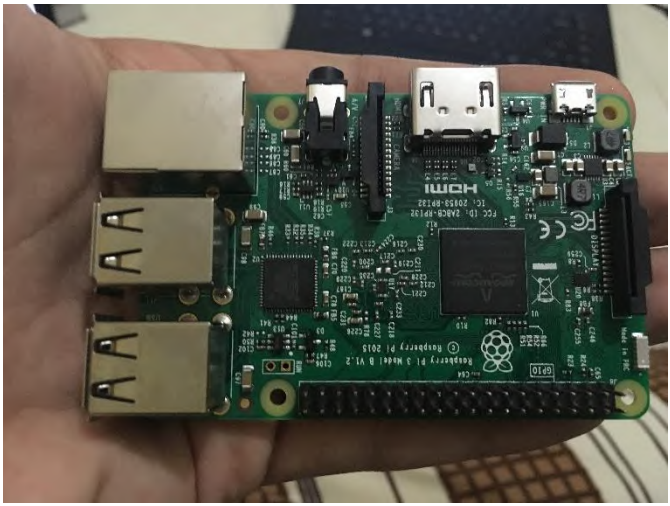
● = Terdapat beberapa fitur yang tidak bisa

2.5.4. OFSoftSwitch

OFSoftswitch [9] adalah perangkat lunak switch OpenFlow yang dikembangkan oleh institut riset CPqD berdasarkan implementasi Ericsson TrafficLab Softswitch. OFSoftswitch mendukung OpenFlow 1.3 terdapat juga versi untuk Linux. Proyek ini juga masih aktif dikembangkan namun memiliki kekurangan dokumentasi dan juga jarang ada penelitian atau papar yang menggunakan OFSoftSwitch.

2.6. Raspberry Pi

Raspberry Pi atau lebih sering disingkat dengan Raspi. Raspberry Pi merupakan salah satu *low cost embedded machine*. Raspberry Pi adalah komputer papan tunggal (*Single Board Circuit* / SBC) besar Raspberry Pi seukuran telapak tangan seperti ynag dapat dilihat pada Gambar 2.9. Raspberry Pi dapat digunakan untuk keperluan uji coba.



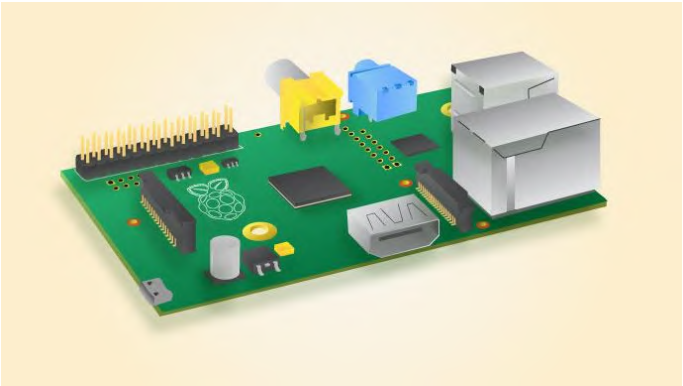
Gambar 2.9 Perbandingan Ukuran Raspi

Pondasi Sistem operasi resmi milik Raspberry Pi adalah Rasbpian. Bahasa yang didukung oleh Raspbian bermacam-macam, utamanya adalah Python, bahasa lainnya juga didukung pada Raspbian seperti C, C++, Java, Perl, Ruby, dan lain sebagainya. Penyimpanan data didesain tidak untuk menggunakan *hard disk* atau *solid-state drive*, melainkan mengandalkan kartu SD (SD *memory card*) untuk *booting* dan penyimpanan jangka panjang. Terdapat tiga distributor Raspberry Pi yaitu Element 14, RS, Alliedelec.

2.6.1. Raspberry Pi 1 Model B

Raspberry Pi 1 model B yang dapat dilihat pada Gambar 2.10, merupakan seri awal dari Raspberry Pi. Model B memiki spesifikasi tertinggi dibandingkan model-model lain pada Raspberry Pi 1. Raspberry memiliki spesifikasi sebagai berikut:

- 700 MHz ARM1176JZF-S Core CPU (ARMv6)
- 512 MB RAM
- Dua USB *port*
- 100 MB *Ethernet port*
- *Camera interface*
- *Display interface*



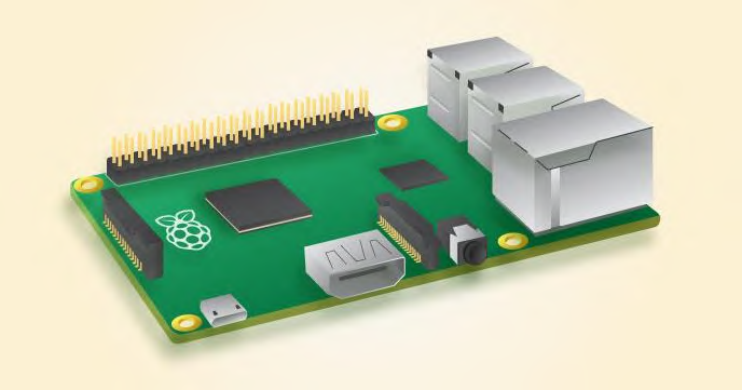
Gambar 2.10 Raspberry Pi 1 Model B

2.6.2. Raspberry Pi 2 Model B

Raspberry Pi 2 model B yang dapat dilihat pada Gambar 2.11 merupakan generasi kedua dari Raspberry Pi. Raspberry Pi 2 rilis pada bulan Febuari tahun 2015. Terdapat beberapa perbedaan baik fisik dan juga fitur. Penambahan fitur pada Raspberry Pi 2 dibandingkan dengan Raspberry Pi 1 antar lain:

- A 900MHz quad-core ARM Cortex-A7 CPU
- 1GB RAM
- 4 port USB

Dikarenakan pada Raspberry Pi 2 model B sudah memiliki prosesor ARMv7, sehingga dapat dijalankan ARM GNU/Linux, termasuk Snappy Ubuntu dan juga Microsoft Windows 10.



Gambar 2.11 Raspberry Pi 2 Model B

2.6.3. Raspberry Pi 3 Model B

Raspberry Pi 3 model B merupakan generasi ketiga Raspberry Pi, menggantikan Raspberry Pi 2. Raspberry Pi 3 rilis pada bulan Febuari 2016. Raspberry Pi 3 secara fisik tidak berbeda dengan Raspberry Pi 2 model B. Adapun beberapa fitur dan perubahan yang dimilik Raspberry Pi 3 dibandingkan seri sebelumnya:

- A 1.2GHz 64-bit quad-core ARMv8 CPU
- 802.11n Wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)

2.7. Permasalahan Lingkungan Uji Coba SDN

Pada saat ini permasalahan untuk melakukan percobaan SDN adalah keterbatasan perangkat keras. Hal tersebut karena SDN memerlukan perangkat jaringan yang mendukung OpenFlow. Sehingga perlu menganti perankat jaringan yang ada, menjadi mendukung OpenFlow. Untuk melakukan hal memerlukan biaya yang tidak kecil.

Beberapa alternatif untuk melakukan percobaan terkait dengan SDN dapat dilakukan pada: (1) Mininet emulator, (2) Net-FPGA, (3) switch yang mendukung OpenFlow. Mininet adalah sistem *emulator* yang dijalankan pada komputer untuk mengemulasikan SDN. Pada Mininet [10] dapat membuat topologi dengan jumlah switch dan juga host dalam jumlah yang banyak tanpa menggunakan perangkat sungguhan. Mininet memiliki dokumentasi yang jelas dan juga halaman wiki. Namun pada Mininet tidak menyediakan lingkungan yang sebenarnya. Kedua adalah Net-FPGA [11], Net-FPGA dapat dimanfaatkan untuk skala yang tidak terlalu besar dapat digunakan sebagai sarana menganalisa dan melakukan percobaan SDN. Permasalahan Net-FPGA adalah harga perangkat yang mahal, kompleksitas dan juga menggunakan bahasa pemograman sendiri. Selain dua solusi yang telah disebutkan terdapat juga beberapa sarana *testbed* SDN yang dimiliki oleh beberapa riset institut seperti “OF@TEIN”, “OFELIA”.

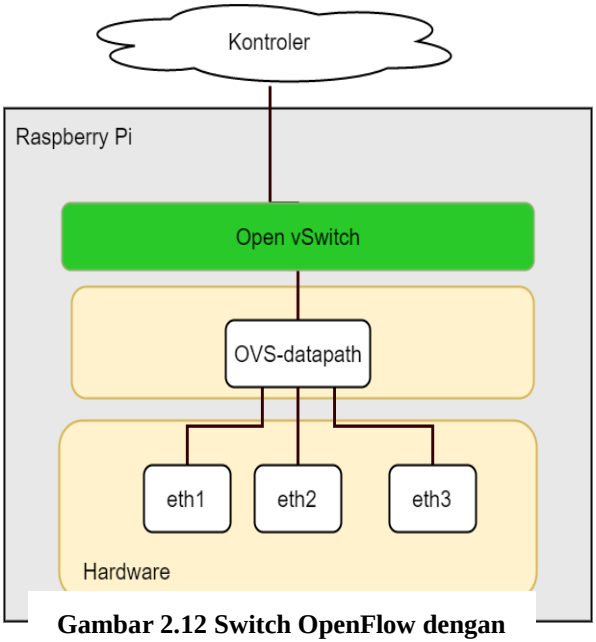
Solusi–solusi tersebut merupakan solusi alternatif untuk melakukan percobaan atau menguji SDN tanpa perlu mengubah infrastruktur jaringan yang ada sekarang menjadi mendukung OpenFlow. Dimana untuk mengubah jaringan menjadi mendukung OpenFlow harus diubah keseluruhan dan perangkat keras jaringan

harus diganti dengan perangkat keras yang mendukung OpenFlow dan membutuhkan biaya.

2.8. Solusi Lingkungan Uji Coba Switch SDN Berbasis Linux

Solusi permasalah lingkungan uji coba pada Tugas Akhir ini dicoba dengan membangun platform SDN dengan menggunkan switch berbasisi Linux. Pada [4] memberikan solusi untuk membuat switch SDN menggunakan Raspberry Pi, perangkat *low-cost embedded* Linux *machine*

Raspberry Pi akan dijadikan switch OpenFlow dengan cara menggunakan perangkat lunak Open vSwitch seperti ynag diilustrasikan pada Gambar 2.12. Open vSwitch merupakan salah satu perangkat lunak switch OpenFlow *open source*.



Gambar 2.12 Switch OpenFlow dengan Open vSwitch

2.9. Kontroler SDN

Selain dibutuhkan switch OpenFlow juga dibutuhkan kontroler untuk mengatur *flow* pada switch. Terdapat banyak pilihan kontroler. Dapat dilihat pada Tabel 2.2 beberapa jenis kontroler yang ada.

Tabel 2.2 Kontroler SDN

Nama kontroler	Antarmuka	Versi OpenFlow
NOX	C++	1.0
POX	Python	1.0
Beacon	Java	1.0
FloodLight	Java, REST	1.0, 1.3
OpenDaylight	REST, OSGi	1.0, 1.3
Ryu	Python, REST, RPC	1.0, 1.2,1.3,1.4

2.9.1. POX/NOX

NOX merupakan kontroler OpenFlow pertama yang dikembangkan oleh Universitas Standford ketika OpenFlow pertama kali dikenalkan. NOX ditulis dengan menggunkan bahasa pemrograman C++ dan Python yang pada akhirnya dipisah digantikan dengan dua kontroler terpisah NOX (C++) dan POX [12] (Python). Banyak *paper* yang mengunakan dua kontroler ini namun NOX dan juga POX sudah tidak lagi dilakukan pengembangan lagi, dapat dilihat dari dukungan versi OpenFlow hanya mendukung OpenFlow 1.0. Untuk mengunduh POX dapat dilakukan dengan mengambil di halaman GitHubnya.

2.9.2. Beacon

Beacon [13] merupakan kontroler berbasis bahasa pemograman Java. Beacon dikembangkan oleh BigSwitch

Network. Pada Beacon sudah terdapat fitur yang langsung dapat digunakan seperti *web-based* GUI, *topologies*. Beacon menggunakan REST API pada antarmuka *northbound.*

2.9.3. OpenDaylight

Open Daylight [14] merupakan kontroler yang sering digunakan. Pada Open Daylight memiliki antarmuka REST, Open Daylight masih aktif dikembangkan dan didukung oleh Linux *Foundation* dan juga Cisco.

2.9.4. Ryu

Ryu [15] merupakan salah satu *controller* SDN *open source* yang cukup populer, sudah digunakan oleh NSA. Ryu menggunakan bahasa pemograman Python. Ryu sudah mensupport OpenFlow versi 1.0, 1.2, 1.3, dan juga ekstensi Nikira. Ryu bersifat *open source* dibawah lisensi Apache 2.0.

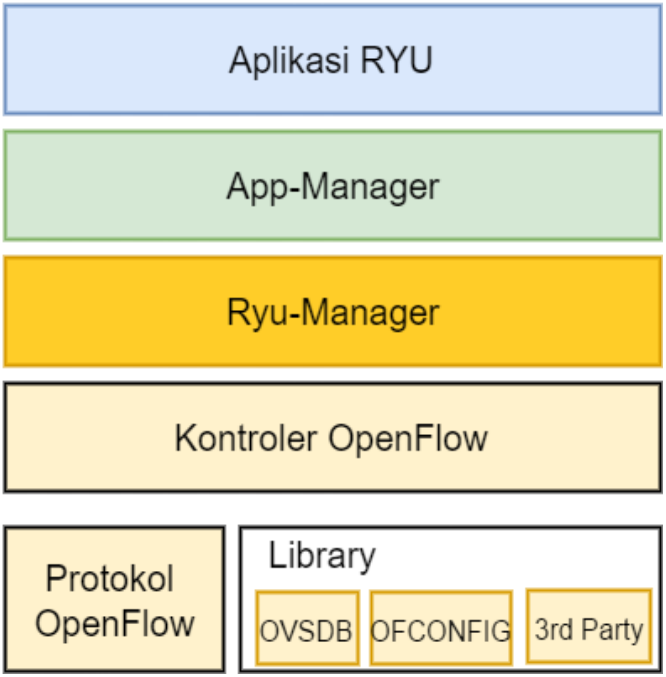
Beberapa kekurangan Ryu dibandingkan kontroler SDN lainya adalah masalah performa, dikarenakan Ryu dibangun menggunakan bahasa pemograman Python. GUI juga tidak terdapat pada controller Ryu. Namun bila dibandingkan dengan kontroler lainnya Ryu memiliki dokumentasi yang lengkap. Pada Ryu terdapat banyak modul *built-in*.

Arsitektur Ryu yang dapat dilihat pada Gambar 2.13 terdiri dari:

1. *Library*,Ryu memiliki banyak *library* untuk mendukung protokol *Southbound* seperti OVSDB dan juga IFCONFIG.
2. Protokol dan juga kontroler OpenFlow, Ryu mendukung protokol OpenFlow 1.4, Kontroler OpenFlow bertanggung jawab mengatur *flow* dari tiap switch
3. Ryu manager secara otomatis akan mendengar pada spesifik IP (contoh: 0.0.0.0) dan port (6633). OpenFlow switch dapat tersambung dengan Ryu manager. Pada IP dan juga port yang telah diatur

4. *App manager* merupakan komponen mendasar dari semua aplikasi Ryu. Semua aplikasi Ryu diturunkan dari kelas App manager.

Pada Ryu sudah terdapat beberapa contoh aplikasi bawaan. Beberapa aplikasi Ryu dapat dijalankan secara bersamaan.



Gambar 2.13 Arsitektur Ryu

2.10. *Routing*

Routing adalah proses untuk memilih jalur yang harus dilalui oleh paket data pada suatu jaringan. Pada umumnya pemilihan rute menggunakan tipe pemilihan jalur terpendek (*the shortest path*). Umumnya pemilihan jalur *routing* yang baik

dibutuhkan dengan memperhitungkan beban jaringan, panjang datagram, *type of service requested* dan pola trafik.
Terdapat 2 jenis routing:

1. *Direct Routing* (*direct delivery*), pola *routing* dengan paket dikirimkan dari satu mesin ke mesin lain secara langsung (host berada pada jaringan fisik yang sama) sehingga tidak perlu melalui mesin lain atau *gateway*.
2. *Indirect Routing (indirect delivery*), pola *routing* dengan paket dikirimkan dari suatu mesin ke mesin yang lain yang tidak terhubung langsung (berbeda jaringan) sehingga paket akan melewati satu atau lebih *gateway* atau jaringan yang lain sebelum sampai ke mesin yang dituju.

Tabel *routing* merupakan kumpulan informasi mengenai jalur yang akan digunakan paket data dalam proses transmisikannya. Informasi yang terdapat pada tabel routing dapat diperoleh secara *static routing* melalui perantara administrator dengan cara mengisi tabel *routing* secara manual ataupun secara *dynamic routing* menggunakan protokol routing, dimana setiap router yang berhubungan akan saling bertukar informasi routing agar dapat mengetahui alamat tujuan dan memelihara tabel routing.Informasi pada tabel *routing* umumnya sebagai berikut:

1. Alamat jaringan tujuan
2. Antarmuka router yang terdekat dengan jaringan tujuan
3. *Metric*, yaitu sebuah nilai yang menunjukkan jarak untuk mencapai jaringan tujuan

BAB III
PERANCANGAN

Pada bab ini dijelaskan mengenai rancangan arsitektur SDN dan juga rancangan arsitektur klaster Hadoop. Perancangan yang dijelaskan meliputi data dan proses perangkat lunak yang digunakan topologi dan juga infrastruktur yang akan dibangun.

3.1. Deskripsi Umum Sistem

Pada Tugas Akhir ini dibangun klaster Hadoop dengan menggunkan arsitektur jaringan SDN. Memanfaatkan arsitektur jaringan SDN, pada klaster Hadoop dilakukan manajemen *transfer rate* untuk dapat mengoptimalkan proses perpindahan data pada saat penyimpanan ke HDFS. Manajemen *transfer rate* dilakukan dengan cara memanfaatkan fitur *queue* pada switch OpenFlow. Tiap *queue* yang dibuat dapat diatur nilai maksimum dan juga nilai minimum *transfer rate.* Dengan menetapkan nilai *transfer rate* pada klaster Hadoop maka lalu lintas untuk proses penyimpanan data ke HDFS terbebas dari *congestion* yang disebabkan dari lalu lintas data lain

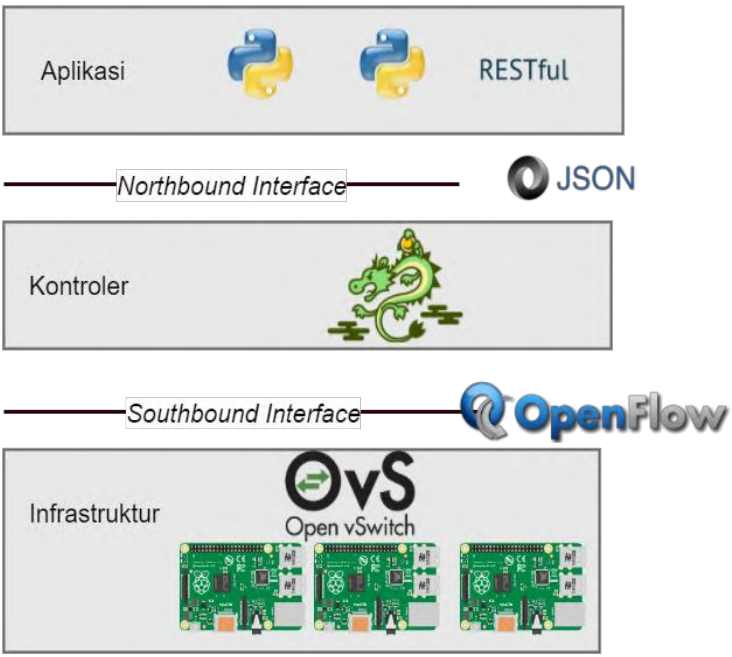
3.2. Lingkungan SDN

Pada Tugas Akhir ini klaster Hadoop akan dibangun pada arsitektur jaringan SDN. Terdapat tiga komponen/layer mendasar untuk mengimplementasikan arsitektur jaringan SDN, sebagai berikut:

- Aplikasi
- Kontroler
- Infrastruktur

Selain itu juga perlu ada antarmuka antar tiap bagian. Antarmuka *Southbound* menghubungkan infrastruktur dan kontroler. *Northbound interface* yang menghubungkan kontroler dengan aplikasi.

Secara umum rancangan komponen SDN yan dibangun pada Tugas Akhir ini dapat dilihat pada Gambar 3.1.



Gambar 3.1 Rancangan Komponen SDN

Seperti yang dapat dilihat pada Gambar 3.1 pada bagian infrastruktur akan digunakan Raspberry Pi dan juga Open vSwitch sebagai switch OpenFlow. Pada bagian kontroler akan digunakan kontroler SDN Ryu, sehingga aplikasi-aplikasi SDN yang digunakan akan menggunkan bahasa pemograman Python.

Pada sub-bab ini akan dijelaskan mengenai rancangan proses yang dilakukan untuk membangun arsitektur SDN, dimulai dari pengaturan Raspberry Pi, rancangan switch OpenFlow, rancangan kontroler SDN, dan juga aplikasi untuk membuat arsitektur SDN.

3.2.1. Pengaturan Raspberry Pi

Raspberry Pi akan digunakan sebagai switch OpenFlow. Permasalahan Raspberry Pi hanya memiliki 1 port ethernet sementara dibutuhkan lebih dari satu port ethernet yang akan digunakan, selain untuk menghubungkan dengan kontroler port ethernet juga dibutuhkan untuk menyambungkan *host*. Hal tersebut dapat disiasati dengan menggunkan adapter USB *to* ethernet.

Gambar 3.2 Diagram Alir Pengaturan Raspberry Pi

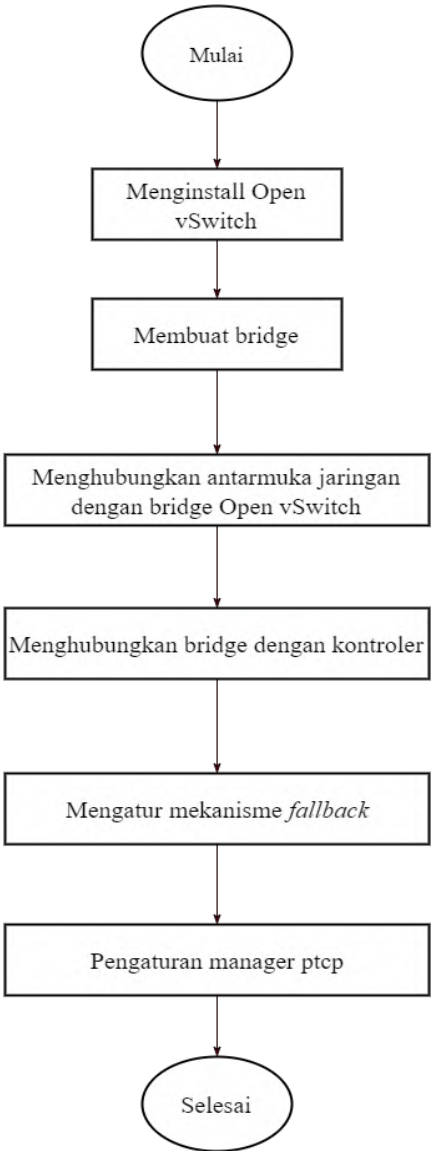
Selain hal tersebut juga beberapa tahapan yang harus dilakukan untuk mempersiapkan Raspberry Pi seperti yang dapat dilihat pada Gambar diagram alir 3.2. Seperti mempersiapkan sistem operasi yang digunakan, mengatur antarmuka jaringan dan juga pengaturan fitur meneruskan paket data pada Raspberry Pi yang akan digunakan.

3.2.2. Switch OpenFlow

Salah satu alternatif selain membeli perangkat switch OpenFlow adalah menggunakan perangkat lunak OpenFlow. Beberapa perangkat dapat diubah menjadi perangkat yang mendukung OpenFlow. Pada [4] memberikan solusi untuk membuat switch SDN menggunakan perangkat *low-cost embedded* Linux *machine* dan juga Open vSwitch. Perangkat *low-cost embedded* Linux *machine* yang digunakan adalah Raspberry Pi.

Terdapat enam langkah untuk mengubah Raspberry Pi menjadi switch OpenFlow seperti yang dapat dilihat pada Gambar digram alir 3.3. Langkah pertama yang dilakukan adalah melakukan instalasi Open vSwitch pada Raspberry Pi. Aplikasi Open vSwitch sudah terdapat pada distro Linux, sehingga untuk menginstall cukup menjalankan perintah *apt-get install*. Pada saat menginstall Open vSwitch akan terinstall juga OVSDB [16] (Open vSwitch *database*). OVSDB adalah antarmuka yang digunakan untuk melakukan manajemen dan juga konfigurasi pada Open vSwitch. Langkah berikutnya adalah membuat bridge. Bridge yang telah dibuat akan dinisialisasikan dengan antarmuka jaringan yang ada pada Raspberry Pi. Port yang dinisialisasikan menjadi port yang mendukung OpenFlow. Setelah melakukan inisialisasi port mana saja yang tersambung pada bridge langkah berikutnya adalah menghubungkan switch OpenFlow dengan kontroler yang ada. Pada Open vSwitch kontroler dan juga bridge berinteraksi lewat port tcp 6633. Pada Open vSwitch terdapat dua mekanisme *fallback*

- *Secure*
- *Stand alone*



Gambar 3.3 Diagram Alir Membuat Switch OpenFlow

3.2.3. Kontroler

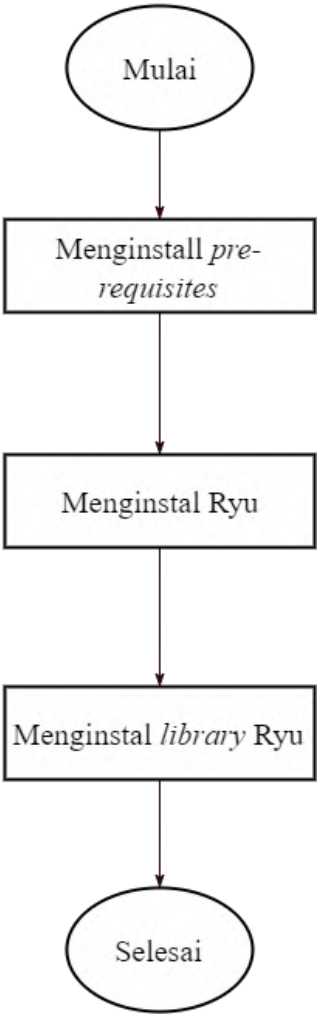
Kontroler yang digunakan pada Tugas Akhir ini adalah Ryu. Pemilihan Ryu sebagai kontroler dikarenakan memiliki dokumentasi yang lengkap dan juga forum yang aktif. Selain itu juga Pada Ryu Terdapat banyak contoh aplikasi yang dapat digunakan.

Fungsi utama kode Ryu terletak pada /ryu/folder(pada komputer penulis terletak pada *(/home/Narendra/ryu/ryu)* terdapat enam komponen pada struktur kode ryu, antara lain:

1. App/, berisi aplikasi yang dapat dijalankan diatas kontroler
2. Base/, berisi kelas dasar dari aplikasi Ryu, kelas RyuApp perlku diturunkan ketika mebuat aplikasi baru
3. Controller/, berisi file untuk menangani fungsi OpenFlow, seperti paket dari switch, mengumpulkan statistic, dan juga menangani *network events*.
4. Lib/, berisi kumpulan paket *library* untuk mengurai *header* protokol dan juga *library* untuk OFConfig.
5. Ofproto/, berisi informasi spesifik cara untuk mengurai berbagai macam versi protokol OpenFlow (1.0,1.2,1.3,1.4)
6. Topology/, berisi kode untuk menampilkan topologi OpenFlow switch.

Untuk menggunkan Ryu sebagai kontroler langkah pertama adalah menginstal beberapa *library* pra-syarat seperti *python-webob*, OVS. Langkah berikutnya adalah melakukan instalasi Ryu, untuk dapat melakukan instalasi Ryu terdapat dua cara, cara pertama dengan melakukan *clone* GitHub aplikasi Ryu atau dengan cara dengan menggunkan *pip install*. Setelah melakukan instalasi Ryu perlu juga dilakukan penambahan *library* pendukung yang dibutuhkan oleh fungsi-fungsi pada Ryu seperti OVS yang dibutuhkan oleh fungsi OVSDB. Untuk menjalankan Ryu dapat dilakukan dengan menjalankan perintah dibawah.

```
ryu-manager namaAplikasi
```



Gambar 3.4 Diagram Alir Instalasi Ryu

3.2.4. Aplikasi

Sub-bab ini membahas bagaimana rancangan dari aplikasi yang akan digunakan dalam arsitektur yang akan dibangun. Digunakan tiga aplikasi SDN untuk mendukung *platform* SDN yang dibangun, aplikasi untuk menghubungkan antar host, *switch hub*, *router* dan juga aplikasi pengaturan *transfer rate*.

3.2.4.1. *Switch Hub*

Untuk model satu switch dibutuhkan aplikasi yang dapat melakukan:

1. Mempelajari alamat fisik dari host yang tersambung pada port yang ada, dan menyimpannya pada tabel alamat fisik.
2. Ketika menerima paket yang alamatnya sudah tercatat, paket akan dikirimkan menuju port yang tersambung dengan host yang dituju.
3. Ketika menerima paket yang alamatnya hostnya belum diketahui, akan melakukan *flooding*.

Ketiga hal tersebut dapat diimpolementaikan dengan OpenFlow, dikarenakan switch OpenFlow dapat menerima beberapa instruksi antara lain:

1. Menuliskan alamat dari data yang diterima atau data yang dikirim pada port yang tersambung.
2. Mengirim paket yang diterima ke kontroler (*packet in)*.
3. Meneruskan paket dari kontroler ke port yang dituju (*packet out*).

Dengan memanfaatkan fungsi yang dimiliki oleh switch OpenFlow, dapat diimpelementasikan dengan cara sebagai berikut:

1. Memanfaatkan fungsi *packet-in* untuk mempelajari alamat fisik.
2. Apabila host sudah diketahui maka akan dijalankan dengan menggunakan fungsi *packet-out* untuk mengirim paket ke alamat tujuan
3. Apabila host masih belum diketahui maka akan dijalan fungsi *packet-out* untuk melakuakn proses *flooding*.

Untuk mengimplementasikan pada simple_switch digunakan beberapa fungsi antara lain:

1. Inisialisasi, pada fungsi ini dilakukan inisialisasi awal sepert versi OpenFlow yang akan digunakan selain itu juga pada fungsi melakukan inisialisai *inheritance* kelas utama Ryu, *ryu.base.app_manager*.
2. *Event handler*, fungsi ini diimplementasikan untuk memberikan respon kepada pesan OpenFlow.
3. *Packet-in*, fungsi ini untuk menangani kejadian-kejadian paket masuk seperti menerima paket yang belum diketahui tujuannya, memperbarui tabel alamat fisik, menentukan port tujuan, menambahkan alur baru pada tabel alamat fisik dan juga meneruskan paket tujuan.

3.2.4.2. Router

Pada Tugas Akhir ini juga akan digunakan aplikasi Ryu router. Aplikasi router pada Ryu dapat melakuakn *routing* secara *single tenant* dan juga *multi tenant*. Aplikasi router pada Ryu merupakan aplikasi RESTful sehingga dapat menerima inputan atau perintah menggunakan JSON.

Tabel 3.1 Daftar API

Fungsi	Metode	Alamat	Data
Mengatur alamat	POST	/router/{swith}	Alamat
Mengatur rute statik	POST	/router/{switch}	Tujuan dan juga Gateway
Mengatur rute *default*	POST	/router/{switch}	gateway
Menghapus alamat	DELETE	/router/{switch}	id alamat
Menghapus rute	DELETE	/router/{switch}	Id rute

Melihat pengaturan router	GET	/router/{switch}	-

3.2.4.3. Pengaturan *Transfer Rate*

Pada Ryu terdapat aplikasi pengaturan *transfer rate*. Aplikasi pengaturan *transfer rate* yang digunakan pada Ryu dilakukan dengan mekanisme membuat pengaturan *queue* dan mendefinisikan nilai maksimum atau minimum transfer rate. Sama seperti aplikasi router aplikasi pengaturan *transfer rate* juga merupakan aplikasi RESTful.

Fungsi		URL	Data
Mendapatkan status *queue*	GET	/qos/queue/status/{switch}	-
Mendapatkan konfigurasi *queue*	GET	/qos/queue/{switch}	-
Mengatur *queue*	POST	/qos/queue/{switch}	- Nama port - Jenis Port - Nilai maks - Nilai min
Menghapus *queue*	DELETE	/qos/queue/{switch}	-
Mendapatkan semua pengaturan *queue*	GET	/qos/rules/{switch}	-
Pengaturan Qos	POST	/qos/rules/{switch}	-Prioritas -Port masuk -Nw_src -Nw_dst -Protokol

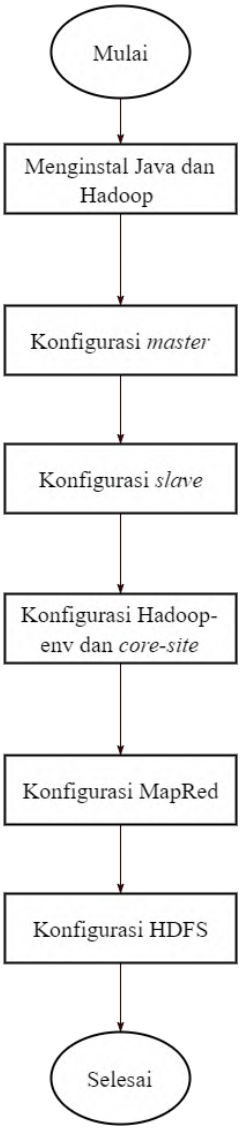
3.3. Hadoop

Pada Tugas Akhir ini akan digunakan Hadoop. Hadoop yang akan digunakan Hadoop versi 1.2. Hadoop menggunakan bahasa pemograman Java sehingga perlu dilakukan instalasi Java pada node yang akan dijalankan Hadoop. Untuk melakukan instalasi dan juga konfigurasi Hadoop dapat dilihat pada Gambar diagram alir 3.5.

Tahap awal dilakukan pengunduhan Hadoop yang dapat diambil pada halaman resmi Hadoop. Langkah berikutnya dilakukan konfigurasi pembagian peran tiap node¸apakah menjadi Datanode atau menjadi Namenode yang diinisialisasikan pada file *master* dan juga *slave*. Tahapan ketiga adalah melakukan konfigurasi Hadoop *core*. Pengaturan dilakukan pada *hadoop-env.sh* berisi tentang konfigurasi lingkungan Hadoop saat berjalan. Seperti Java yang dipakai oleh Hadoop saat mengeksekusi program, ukuran *heap* dan sebagainya. Setelah melakukan konfigurasi Hadoop-*core* tahapan berikutnya adalah melakukan konfigurasi Hadoop mapred yang dilakukan pada file *mapred-site.xml*. Konfigurasi *mapre-site.xml* untuk Namenode paling penting adalah memberi nama hostname namenode dan port yang akan dialokasikan. Untuk Datanode konfigurasi *mapred.job.tracker* harus sama seperti *nilai mapred.job.tracker* pada Namenode-nya. Tahapan terakhir adalah melakukan konfigurasi HDFS yang dapat dikonfigurasikan pada file HDFS. HDFS menangani seluruh data yang disimpan pada Hadoop. Konfigurasi penting untuk HDFS jumlah replikasi data, *path* Namenode, dan *path* Datanode pada sistem yang bersifat lokal.

3.3.1. Data Masukan Hadoop

Data yang akan digunakan adalah data berupa teks, dokumen, xml, atau semua dapat yang dapat yang dapat diubah menjadi teks. Besar data yang diuji adalah dari ukuran 1MB, 10MB, dan 20MB yang terdiri dari beberapa data.



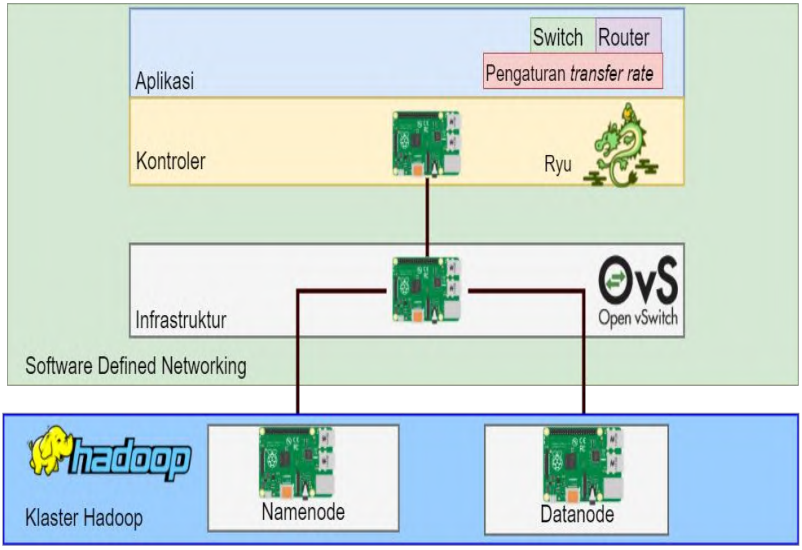
Gambar 3.5 Diagram Alir Instalsi Hadoop

3.4. SDN dan Hadoop

Pada sub-bab ini akan dijelaskan bagaimana topologi klaster Hadoop yang akan dibangun pada arsitektur jaringan SDN yang akan digunakan. Selain itu juga perancangan manajemen *transfer rate* yang bertujuan untuk meningkatkan proses HDFS Hadoop.

3.4.1. Topologi Hadoop dan SDN

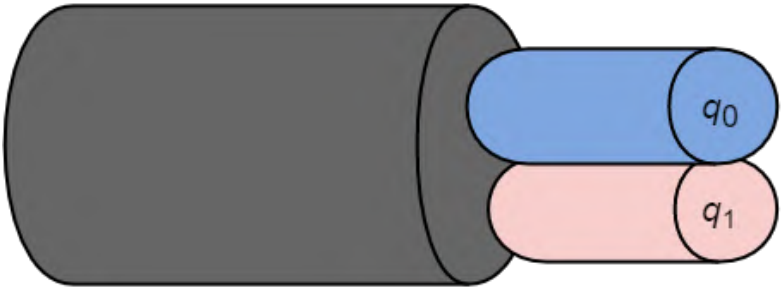
Klaster Hadoop memiliki arsitektur *master-slave* dan SDN memiliki tiga komponen dalam arsitekturnya. Switch OpenFlow akan menjadi switch penghubung antara *master* dan juga *slave* pada klaster Hadoop seperti yang dapat dilihat pada Gambar 3.6. Pada switch OpenFlow akan terhubung Datanode dan juga Namenode. Datanode dan juga Namenode akan terhubung dengan port OpenFlow. Switch OpenFlow akan terhubung dengan kontroler SDN melalui bukan port OpenFlow. Selain topologi satu switch akan dicoba juga model dua switch.



Gambar 3.6 Topologi Hadoop dan SDN

3.4.2. Rancangan Manajemen *Transfer Rate*

Pada kaster Hadoop yang dibangun akan dilakukan manajemen *transfer rate* sehingga proses HDFS Hadoop dapat dioptimalkan. Manajemen *transfer rate* dilakukan dengan cara memanfaatkan fitur yang dimiliki switch OpenFlow. Switch OpenFlow memeiliki fitur pengaturan *transfer rate* dengan menggunakan mekanisme *queue*. *Queue* untuk proses penyimpanan data dipisahkan dengan lalu lintas jaringan yang lain sehingga ketika terjadi *congestion* pada jaringan proses perpindahan data HDFS tidak terganggu. Ilustrasi pengaturan *queue* dapat dilihat seperti pada Gambar 3.7. Dibuat dua buah *queue*, *queue* pertama digunakan untuk proses HDFS Hadoop dan *queue* kedua untuk lalu lintas jaringan tambahan. Pembuatan lalu lintas tambahan untuk membuat *congestion* pada Tugas Akhir ini akan menggunakan Iperf, yang akan mengirimkan paket TCP. Untuk proses HDFS akan diber nilai *transfer rate* yang lebih tinggi dibandingkan nilai *transfer rate* yang akan dihasilkan oleh Iperf.



Gambar 3.7 Pengaturan *Queue*

3.5. Langkah Pengerjaan

Secara umum langkah langkah yang dilakukan dalam penelitian ini dapat dilihat pada Gambar 3.8.

3.5.1. Perancangan Topologi dan Infrastruktur

Pada tahapan ini dilakukan peranacangan dan topologi *platform* yang akan dibangun. Perancangan dilakukan baik perancangan fisik dan juga logikal. Pada tahap perancangan topologi, beberapa pertimbangan yang dilakuakan adalah rancangan harus dapat menjalankan program hadoop sesuai dengan arsitektur yang dibutuhkan Hadoop, dikarenakan sistem nantinya akan menjalankan Hadoop. Selain itu juga topologi juga dirancang sesuai konsep SDN yang memisahkan antara *control plane*, dan juga *data plane*.

3.5.2. Penentuan Kebutuhan

Penentuhan kebutuhan dilakukan setelah melakukan perancangan topologi dan infrastruktur. Pada proses penentuan kebutuhan meliputi pemilihan perangkat keras berdasarkan spesifikasi yang dibutuhkan tiap *tools* yang akan digunakan dan mendata kebutuhan perangkat keras pendukung untuk mendukung rancangan topologi yang akan dibangun.

3.5.3. Instalasi dan konfigurasi OpenFlow

Pada tahapan ini dilakukan instalasi dan juga konfigurasi perangkat keras yang akan dijadikan sebagi Open vSwitch. Pada Tahapan ini adalah awal pengimplemtasian perancangan topologi, dan juga membangun infrastruktur yang mendukung OpenFlow.
Langkah ini bertujuan sebagai syarat dari infrastruktur jaringan SDN yang memerlukan switch yang mendukung OpenFlow yang membutukan interface komunikasi dengan kontroler, *Southbound interface*. Setelah konfigurasi dan juga instalasi berhasil maka akan dilanjutkan pada proses berikutnya.

3.5.4. Instalasi dan konfigurasi SDN

Setelah perangkat switch yang sudah mendukung OpenFlow telah siap digunakan tahap berikutnya adalah instalasi dan juga konfigurasi SDN. Pada tahapan ini dilakukan pemilihan kontroller, instalasi kontroler, mengimplementasikan aplikasi SDN yang akan digunakan dan juga menghubungkan switch dengan kontroler, sehingga kontroler dapat memberikan instruksi yang sesuai untuk SDN *datapath,* melalui OpenFlow yang sesuai dengan aplikasi inginkan.

3.5.5. Instalasi dan Konfigurasi Hadoop

Pada proses ini selain melakukan instalasi dan juga konfigurasi Hadoop, pembagian peran *master node* dan juga *slave node* akan dikonfigurasikan pada tiap node sesuai dengan topologi yang dirancang sebelumnya.

3.5.6. Melakukan Manajemen *Transfer Rate*

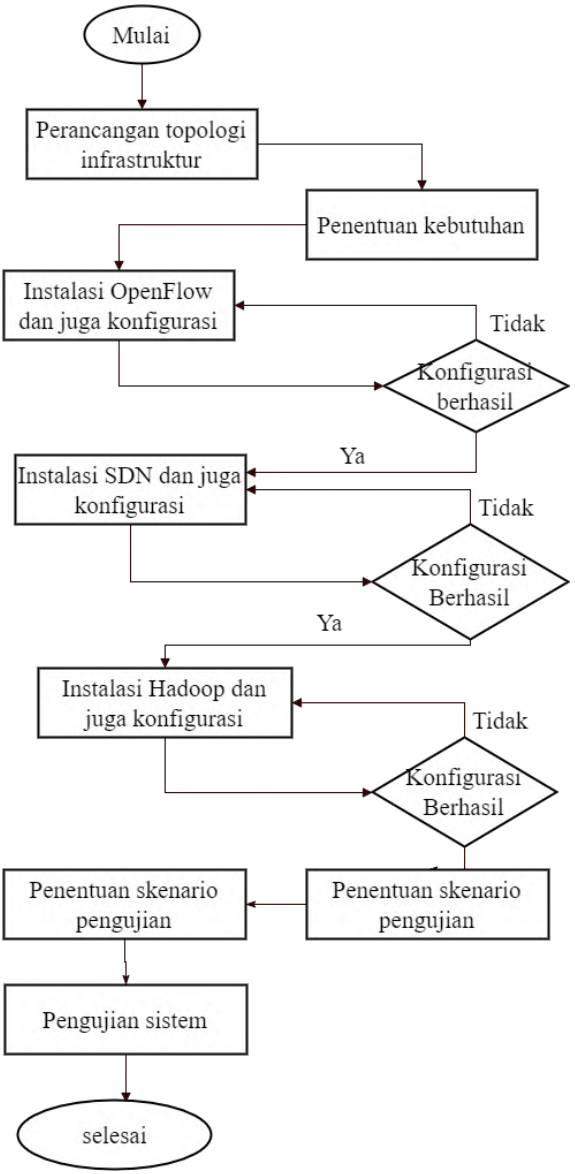
Pada proses ini dilakukan manajeman *transfer rate* sehingga dapat mengoptimalkan proses HDFS Hadoop. Pada proses ini akan dibuat *queue* yang memisahkan lalu lintas jaringan HDFS dan lalu lintas jaringan lainnya.

3.5.7. Penentuan dan Perancangan Skenario

Pada tahap ini dilakukan perancangan skenario untuk melakuakn pengujian. Seperti jumlah node penngubahan *bandwith*¸ dan juga besar file yang akan dieksekusi.

3.5.8. Pengujian Sistem

Pada tahap ketujuh pengujian sistem, sistem yang dibangun akan diuji sesuai dengan skenario yang telah dirancang pada tahap penentuan skenario pengujian. Pada tahap ini pengujian akan dilakukan dengan cara memjalankan program aplikasi SDN dan juga proses penyimpanan ke HDFS.



Gambar 3.8 Langkah Pengerjaan

[Halaman ini sengaja dikosongkan]

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya.

4.1. Lingkungan Implementasi

Pada Tugas Akhir ini lingkungan untuk membangun sistem yang dibangun terdiri atas perangkat keras dan perangkat lunak. Adapun perangkat keras dan juga perangkat lunak yang digunakan dapat dilihat pada Tabel 4.1.

Tabel 4.1 Lingkungan Implementasi Perangkat Lunak

Perangkat Keras	- Raspberry Pi 2 Model B - Raspberry Pi 3 Model B - Komputer dengan Prosesor Intel® Core™ 2 Duo P870 CPU2.53 GHz dengan RAM 2 GB
Perangkat Lunak	Sistem Operasi : - Ubuntu - Raspbian Perangkat Pengembang : - Open vSwitch - Hadoop - Ryu - Atom

4.2. Implementasi Arsiterktur SDN

Sub-bab ini membahas mengenai implementasi arsitektur SDN yang meliputi implementasi pengaturan Raspberry Pi, implementasi pembuatan Open vSwitch, implementasi kontroler SDN dan juga implementasi aplikasi SDN yang dibangun.

4.2.1. Implementasi Pengaturan Raspberry Pi

Pada tahapan ini akan dilakukan pengaturan Raspberry Pi, sehingga Raspberry Pi dapat digunkan sebagai Open vSwitch. Terdapat empat tahapan dalam proses implementasi ini.

4.2.1.1. Mengunduh Raspberry Pi

Tahap pertama yang dilakukan pada pengaturan Raspberry Pi adalah mengunduh sistem operasi Raspberry Pi. Sistem operasi yang dipilih adalah Raspbian. Sistem operasi Raspbian dapat diunduh pada halaman resmi Raspberry Pi. Pada komputer Linux dapat dilakukan dengan cara menjalankan perintah sebagai berikut:

```
% wget https://downloads.raspberrypi.org/
raspbian_lite_latest
```

4.2.1.2. *Writing Image* Raspbian Kedalam SD *card*

Tahap kedua dalam mempersiapkan Raspberry Pi adalah melakukan *write* sistem operasi yang telah diunduh kedalam SD *card* sehingga dapat digunakan oleh Raspberry Pi. Untuk melakukan hal tersebut dapat dilakukan dengan cara menjalankan perintah sebagai berikut:

```
% dd if=2016-05-27-raspbian-jessie.img \
of=/dev/mmcblk0 bs=4M
```

4.2.1.3. Mengatur Antarmuka Jaringan

Tahap ketiga adalah melakukan pengaturan antarmuka jaringan. Untuk melakukan melakukan pengaturan antarmuka jaringan perlu terlebih dahulu melakukan *login* ke Raspberry Pi. *Login* pada Raspberry Pi untuk pertama kali dapat dilakukan dengan *username* dan juga sandi bawaan dari Raspbian yaitu *username*: pi dan sandi: raspberry. Pengaturan antarmuka jaringan

dapat dilakukan dengan cara menuliskan konfigurasi pada file */etc/network/interfaces* yang dapat dilakukan dengan cara memasukan perintah sebagai berikut:

```
% sudo nano /etc/network/interfaces
```

Untuk memberikan alamat IP secara statis pada eth0 tambahkan konfigurasi alamat IP pada file *interfaces*, pastikan alamat IP yang diberikan tiap Raspberry Pi memiliki berbeda, sebagai berikut:

```
autho eth0
iface eth0 inet static
address 192.168.1.31
```

Untuk menambah antarmuka jaringan pada Raspberry Pi dapat digunakan *adapter* USB *to* ethernet pada port USB yang dimiliki oleh Raspberry Pi seperti yang dapat dilihat pada Gambar 4.1 dan 4.2 sehingga menjadi seperti yang dapat dilihat pada Gambar 4.3.

Gambar 4.1 *Adapter* USB *to* Ethernet

Gambar 4.2 Port USB Raspberry Pi

Gambar 4.3 Raspberry Pi dengan Antarmuka Jaringan Tambahan

Untuk melihat pemetaan susunan port ethernet yang dibentuk dapat dilihat dengan menjalankan perintah sebagai berikut:

```
% nano /etc/udev/rules.d/70-persistent-net.rules
```

```
#USB to ethernet adapter
SUBSYSTEM=”net”, ACTION==”add”, DRIVERS=”?*”,
ATTR{address}==”{mac address perangkat}”,
ATTR{dev_id}==”0x0”, ATTR{type}==”1”,
KERNEL==”eth*”, NAME=”eth{nomer eth}”
```

Pada pengerjaan Tugas Akhir ini terdapat permasalahan alamat fisik *adapter* ethernet memiliki alamat fisik yang sama, untuk menyelesaikan permasalahan tersebut dilakukan pengaturan alamat fisik pada pengaturan *interfaces* yang dapat dilakukan dengan menambahkan konfigurasi sebagai berikut:

```
## memberikan alamat fisik
Allow-hotplug eth1
Iface eth1 inet dhcp
Hwaddress ether 00:00:00:00:00:01
```

4.2.1.4. Mengaktifkan Fitur IPv4 *Forwarding*

Tahap ke-empat adalahmengaktifkan fitur menuruskan paket IPv4 pada perangkat Raspberry Pi, untuk melakukan hal tersebut dilakukan dengan cara mengubah pada pengaturan pada file sysctl.conf yang dapat dilakukan dengan menjalankan perintah sebagai berikut:

```
% sudo nano /etc/systl.conf
```

Sebelum:

```
# Uncomment the next line to enable packet
forwarding for IPv4
#net.ipv4.ip_forward=1
```

Sesudah

```
# Uncomment the next line to enable packet
forwarding for IPv4
net.ipv4.ip_forward=1
```

Untuk menguji fitur *forwarding* dapat dilakukan dengan cara menjalankan perintah sebagai berikut:

```
% sysctl net.ipv4.ip_forward
net.ipv4.ip_forward=1
```

4.2.2. Implementasi Switch OpenFlow

Pada tahapan ini akan dilakukan pembuatan switch OpenFlow menggunkan Raspberry Pi dan juga Open vSwitch. Terdapat enam tahapan untuk membuat Raspberry Pi menjadi dapat digunakan menjadi switch OpenFlow. Open vSwitch mengambil alih fungsi fungsi *switching* dari kernel Linux pada layer 2. Open vSwitch telah tersedia pada paket Raspbian dengan versi 2.3.0.

4.2.2.1. Instalasi Open vSwitch

Tahap pertama adalah melakukan instalasi Open vSwitch. Instalasi Open vSwitch dapat dilakukan dengan cara menjalankan perintah sebagi berikut:

```
% sudo apt-get update
% sudo apt-get install openvswitch-switch
% sudo apt-get install bridge-utils
```

Setelah menjalankan perintah instal, untuk menguji apakah instalasi berhasil, dapat dijalankan perintah sebagi berikut:

```
% ovs-vsctl show
```

pada akhir keluaran program akan menampilkan ID, yang otomatis dibuat oleh Open vSwitch, yang merepresentasikan *datapath*.

Pada proses instal Open vSwitch akan ikut juga terinstal juga Open vSwitch *database* (OVSDB). OVSDB adalah manajemen protokol dalam lingkungan SDN. Untuk menguji apakah komponen OVSDB juga terinstal dapat dilakukan dengan menjalankan perintah *ps grep*, apabila OVSDB juga terinstal makan akan menampilkan keluaran *ovsdb-server* dan juga *ovs-vswitchd*, sebagi berikut:

```
% ps –e | grep ovs
```

Langkah berikutnya adalah melakukan konfigurasi Open vSwitch, terdapat beberapa konfigurasi yang harus dilakukan, karena bila tidak dilakukan konfigurasi Open vSwitch tidak memiliki fungsi. Terdapat empat program *command line* untuk melakukan konfigurasi Open vSwitch.

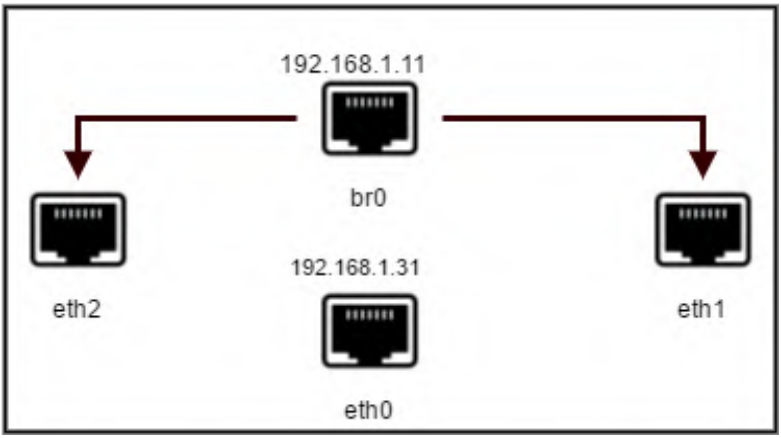
- *Ovs-vsctl*, untuk melakukan konfigurasi *ovs-vswitchd*, seperti pembuatan *bridge*, menyambungkan ke kontroler.
- *Ovs-oftcl*, komunikasi dengan bagian OpenFlow.
- *Ovs-appctl*, *tool* untuk vSwitch daemon
- *ovs-dptcl*, mengatur tabel alur (*flow table*)

4.2.2.2. Membuat Bridge

Tahap kedua dalam rangkain proses membuat switch OpenFlow adalah membuat *bridge* Open vSwitch. Konfigurasi membuat *bridge* untuk semua antarmuka yang tergabung dengan jaringan OpenFlow adalah hal yang terpenting. Untuk membuat *bridge* dapat dilakukan dengan menjalankan perintah sebagi berikut:

```
% ovs-vsctl add-br br0
```

Apabila *bridge* berhasil dibentuk dapat dilihat dengan menjalankan perintah *ifconfig* dan juga perintah *ovs-vsctl show. Bridge* padaOpenFlow dapat diilustrasikan seperti pada Gambar 4.4.



Gambar 4.4 Ilustrasi *Bridge* OpenFlow

Seperti yang dapat dilihat pada Gambar 4.4 semua antarmuka yang tersambung dengan *bridge* br0 akan menjadi antarmuka OpenFlow. Pada Tugas Akhir ini antarmuka jaringan yang menggunakan *adapter* ethernet akan tersambung dengan OpenFlow *bridge* sehingga menjadi antarmuka OpenFlow, sementara eth0 terhubung dengan kontroler sehingga tidak dijadikan antarmuka OpenFlow.

4.2.2.3. Menghubungkan Antarmuka Jaringan dengan Bridge

Setelah bridge berhasil dibuat langkah berikutnya adalah menghubungkan antarmuka jaringan dengan *bridge* OpenFlow yang telah dibuat. Untuk menyambungkan antarmuka dengan *bridge* OpenFlow dapat dilakan dengan cara menjalankan perintah sebagai berikut:

```
% ovs-vsctl add-port bro eth1
% ovs-vsctl add-port br0 eth2
```

Setelah melakukan tersebut langka berikutnya adalah menjalankan perintah sebagai berikut:

```
%  ifconfig br0 192.168.1.11 \
   netmask 255.255.255.0 up
% ifconfig eth1 0 up
% ifconfig eth2 0 up
```

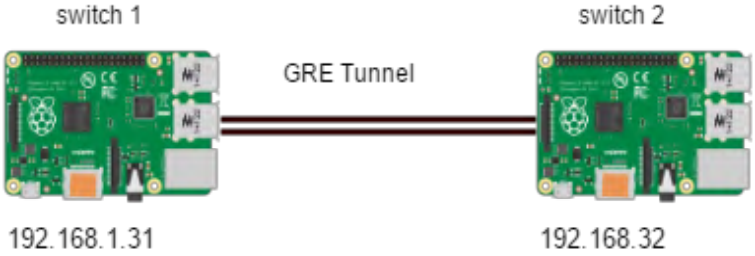
Antarmuka yang terhubung tidak hanya antarmuka fisik, pada Open vSwitch terdapat fitur GRE-*tunnel*. Fitur GRE – *tunnel* dapat digunakan untuk menghubungkan dua switch, untuk menghubungkan dua switch dengan GRE –*tunnel* dapat dilakukan dengan cara menjalankan perintah
Pada switch 1:

```
% ovs-vsctl add-port ovsbr0 gre1 – set interface
gre1 type=gre \
options:remote_ip:192.168.1.32
```

Pada switch 2:

```
% ovs-vsctl add-port ovsbr0 gre1 – set interface
gre1 type=gre \
options:remote_ip:192.168.1.31
```

Pada [4] digunakan metode GRE Tunnel untuk menyambungkan kedua switch seperti yang dapat dilihat pada gambar 4.5 Pada saat menggunakan GRE- tunnel yang perlu diperhatikan adah ukuran dari MTU pada tiap host yang terhubung dengan switch. Ukuran standar MTU adalah 1500, untuk mengecek ukuran MTU dapat dilakukan dengan menjalankan perintah *ifconfig* dan nama port.



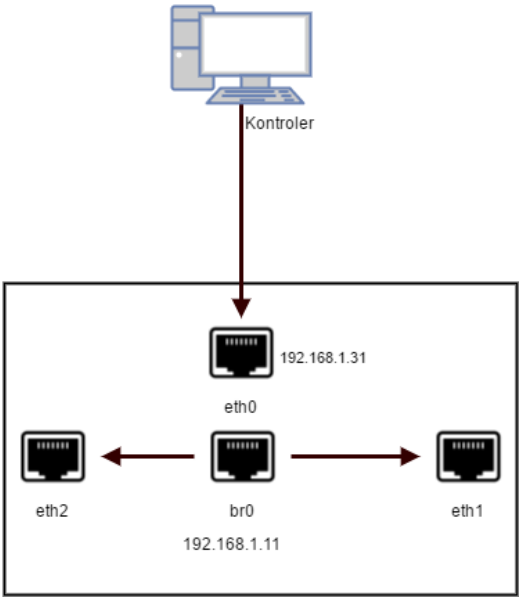
Gambar 4.5 GRE-Tunnel

4.2.2.4. Menghubungkan Bridge dengan Kontroler

Tahap ke-empat setelah tiap antarmuka sudah tersambung dengan *bridge* adalah menghubungkan *bridge* dengan kontroler sehingga kontroler dan juga switch dapat saling berkomunikasi. *Bridge* dan juga kontroler saling berkomunikasi menggunakan protokol TCP menggunkan *port* 6633. Cara menghubungkan *bridge* dan juga kontroler dapat dilakukan dengan menjalankan perintah sebagai berikut:

```
% ovs-vsctl set-controller br0 tcp:192.168.1.10:6633
```

Bridge dan juga kontroler terhubung bukan pada port OpenFlow. Seperti yang dapat dilihat pada Gambar 4.6 kontroler dan juga switch terhubung pada port eth0, yang bukan merupakan port OpenFlow seperti port eth1 dan juga port eth2.



Gambar 4.6 Bridge dan Kontroler

4.2.2.5. Mengatur Mekanisme Fallback

Tahap kelima setelah *bridge* tersambung dengan kontroler tahap adalah mengatur mekanisme *fallback*. Mekanisme *fallback* pada Open vSwitch terjadi apabila tidak mendapatkan respon dari kontroler setelah melakukan tiga kali percobaan sambungan. Pada Open vSwitch terdapat dua mekanisme *fallback* yaitu:

- *Secure*
- *Standalone*

Untuk menjalankan mekanisme *fallback* dapat dilakukan dengan menjalankan perintah sebagai berikut:

```
% ovs-vsctl set-fail-mode br0 secure
```

4.2.2.6. Mengatur Manager

Tahapan ke-enam adalah melakukan pengaturan *manager* pada switch OpenFlow. Pengaturan *manager* diperlukan untuk nantinya digunkan memasukan konfigurasi kedalam OVSDB, dengan melakukan konfigurasi *manager*, kontroler nantiny dapat memasukan konfigurasi dan juga pengaturan ke OVSDB sesaui dengan port yang telah dinisialisasikan. Untuk melakukan pengaturan *manager* dapat dilakukan dengan menjalankan perintah sebagai berikut:

```
% ovs-vsctl set-manager ptcp:6632
```

4.2.2.7. Tahap Pengecekan

Setelah semua tahapan konfigurasi Open vSwitch dilakukan. Tahapan berikutny adalah melakukan pengecekan konfigurasi sperti id *datapath*, nama *bridge*, IP dari kontroler, dan antarmuka jaringan mana saja yang tersambung seperti yang dapat dilihat pada Gambar 4.7, untuk melakukan hal tersebut dapat dilakukan dengan cara menjalankan perintah sebagai berikut:

```
% ovs-vsctl show
```

```
pi@raspberrypi:~ $ sudo ovs-vsctl show
3afddf15-1ece-4a0f-b1f2-778ca4484c91
    Manager "ptcp:6632"
    Bridge "s2"
        Controller "tcp:192.168.1.10:6633"
        Port "eth2"
            Interface "eth2"
        Port "eth1"
            Interface "eth1"
        Port "s2"
            Interface "s2"
                type: internal
```

Gambar 4.7 Pengecekan Konfigurasi

4.2.3. Implementasi Kontroler

Pada tahapan ini membahas implementasi kontroler yang digunakan pada platform SDN yang dibangun. Keluaran dari implementasi ini agar dapat mengatur *flow table* pada switch OpenFlow dan juga memenuhi 3 komponen SDN. Adapun tahapannya dibagi menjadi 3 tahapan. Yaitu instalasi pra-syarat, instalasi Ryu dan juga instalasi *library* pendukung Ryu.

4.2.3.1. Menginstal Pra-syarat

Sebelum melakukan instalasi Ryu, perlu dilakukan beberapa instalsi pra-syarat yang harus dilakukan agar program Ryu dapat berjalan,antara lain:

1. Git
2. *Python-dev*
3. *Python-webob*
4. *Python eventlet*
5. *Pyhton routes*
6. *Python Paramiko*

Instalasi paket diatas dilakukan dengan menjalankan perintah sebagai berikut:.

```
% sudo apt-get install python-pip python-dev
  python-eventlet python-routes python-webob
  python-paramiko
```

4.2.3.2. Menginstal Ryu

Pada sub-bab ini dijelaskan cara melakukan instalasi kontroler Ryu. Instalasi Ryu cukup mudah dilakukan. Terdapat dua cara untuk melakukan instalasi Ryu:

1. Cara pertama dapat dilakukan dengan menggunakan pip, dengan cara menjalankan perintah sebagi berikut:

```
% pip install ryu
```

2. Cara kedua dapat dilakukan dengan melakukan *clone* aplikasi Ryu dari halaman GitHub, dengan cara menjalankan perintah sebagi berikut:

```
% git clone git://github.com/osrg/ryu.git
% cd ryu; python ./setup.py install
```

Setelah Ryu terinstal, Ryu dapat diakses di komputer kontroler dengan direktori /home/{nama user}/ryu/ryu. Akan terbentuk enam folder antara lain:

1. *App*/, berisi aplikasi yang dapat dijalankan diatas kontroler
2. *Base*/, berisi kelas dasar dari aplikasi Ryu, kelas RyuApp perlu diturunkan ketika mebuat aplikasi baru
3. *Controller*/, berisi *file* untuk menangani fungsi OpenFlow, seperti paket dari switch, mengumpulkan statistic, dan juga menangani *network events*.
4. *Lib*/, berisi kumpulan paket *library* untuk mengurai *header* protokol dan juga *library* untuk OFConfig.
5. *Ofproto*/, berisi informasi spesifik cara untuk mengurai berbagai macam versi protokol OpenFlow (1.0,1.2,1.3,1.4)
6. *Topology*/, berisi kode untuk menampilkan topologi OpenFlow switch.

Menjalankan Ryu dapat dilakukan dengan menjalankan perintah sebagai berikut:

```
% ryu-manager namaAplikasi
```

Untuk menulis aplikasi pada Ryu dapat menggunkan bahasa pemograman Python dan untuk menjalankan aplikasi yang telah dibuat dapat dilakukan dengan menjalankan perintah sebagai berikut:

```
cd ryu
python ./setup.py install
```

4.2.3.3. Menginstal Library Ryu

Tahap ketiga adalah melakukan instalasi *library* tambahan yang dibutuhkan untuk mendukung aplikasi Ryu. Pada pengerjaan Tugas Akhir ini dilakukan instalasi dua paket tambahan yaitu:

1. Lxml, yang dapat dilakukan dengan menjalankan perintah sebagai berikut:

```
% pip install lxml
```

2. OVS, yang dapat dilakukan dengan menjalankan perintah sebagai berikut:

```
% pip install ovs
```

4.2.4. Implementasi Aplikasi Ryu

Pada sub-bab ini akan dijelaskan implementasi dari aplikasi Ryu yang digunakan pada pengerjaan Tugas Akhir ini. Tujuan dari penggunaan aplikasi untuk melakukan pengaturan pada arsitektu jaringan SDN yang dibangun. Terdapat tiga aplikasi yang digunakan yaitu, Switch, Router dan juga pengaturan *transfer rate*.

4.2.4.1. Switch

Aplikasi ini digunakan untuk menghubungkan antar host dengan switch. Dalam aplikasi ini akan meneruskan paket ke kontroler SDN (*packet-in)*, melanjutkan paket dari kontroler SDN ke port yang dituju (*packet-out*) dan juga mencatat pada *flow table* pada switch OpenFlow. Aplikasi ini diimplementasikan dengan menggunakan aplikasi Ryu *simple_switch13*.py. Untuk menjalankan aplikasi dapat dilakukan dengan menjalankan perintah sebagai berikut:

```
% ryu-manager ryu.app.simple_switch_13
```

Pada kode aplikasi Ryu untuk mengatasi paket yang masuk dimplementasikan pada fungsi *packet_in_handler*, pada fungsi *packet_in_handler* terdapat proses memperbarui tabel alamat fisik, menentukan port tujuan, dan mengirimkan paket. Sementara untuk menambahkan alur diimplemetasikan pada fungsi *add_flow*.

4.2.4.2. Router

Aplikasi ini digunakan untuk menghubungkan antar host dengan switch yang memiliki beda subnet. Aplikasi ini diimplementasikan dengan menggunakan aplikasi Ryu *rest router*. Aplikasi *rest router* dapat menerima inputan data berupa JSON. Untuk menjalankan aplikasi dapat dilakukan dengan menjalankan perintah sebagai berikut:

```
% ryu-manager ryu.app.rest_router
```

Pada aplikasi Router untuk untuk menambahkan dan juga menghapus rute dapat dilakukan dengan mengirimkan JSON konfigurasi ke kontroler. Untuk menjalankan aplikasi Router terdapat beberap tahapan, sebagai berikut:

1. Menjalankan perintah pada kontroler dengan perintah:

```
% ryu-manager ryu.app.rest_router
```

2. Setelah menjalankan aplikasi tahap berikutnya adalah melakukan pengaturan alamat IP tiap antamuka pada router, dapat dilakukan dengan cara menjalankan perintah:

```
% curl -X POST –d '{"address":"{ip}"}'
http://localhost:8080/router/{switch}
```

3. Setelah tiap router memilki alamat, dilakukan pengaturan rute, dengan menjalankan perintah:

```
% curl -X POST –d '{"gateway":"{ip}"}'
http://localhost:8080/router/{switch}
```

4.2.4.3. Pengaturan *Transfer Rate*

Aplikasi ini digunakan untuk melakukan manajemen *transfer rate*. Pengaturan *transfer rate* dilakukan dengan cara membuat *queue*. Tipa *queue* akan memiliki nilai *transfer* maksimum dan juga nilai minimum. Setelah dibuat *queue*, tiap *queue* akan dipetakan ke *flow* pada tiap switch, dengan cara menentukan port tujuan dan juga IP tujuan yang akan masuk kedalam *queue* tersebut. Aplikasi ini diimplementasikan dengan menggunakan aplikasi Ryu *QoS*. Aplikasi *QoS* dapat menerima inputan data berupa JSON. Untuk menjalankan aplikasi dapat dilakukan dengan menjalankan perintah sebagai berikut:

```
% ryu-manager ryu.app.rest_qos
ryu.app.qos_simple_switch_13
ryu.app.rest_conf_switch
```

Pada aplikasi pengaturan *transfer rate* untuk untuk menambahkan dan juga menghapus rute dapat dilakukan dengan mengirimkan JSON konfigurasi ke kontroler. Untuk menjalankan aplikasi *transfer rate* terdapat beberap tahapan, sebagai berikut:

1. Pertama menjalankan program QoS dapat dilakukan dengan menjalankan perintah pada kontroler:

   ```
   % ryu-manager ryu.app.rest_qos \
   ryu.app.qos_simple_switch_13 \
   ryu.app.rest_conf_switch
   ```

2. Langkah berikutnya adalah membuat *queue*. Membuat *queue* berisi parameter nilai transfer maksimal dan nilai transfer minimum, dengan perintah dibawah ini:

   ```
   % curl -X POST -d '{"port_name":
   "[port]", "type": "linux-htb",
   "max_rate": "[nilai maksimal]", "queues":
   [{"max_rate": ""}, {"min_rate": ""}]}'
   http://localhost:8080/qos/queue/{switch}
   ```

3. Setelah *queue* berhasil dibuat langkah berikutnya adalah memetakan *queue* ke alamat dan juga port yang spesifik

```
curl -X POST -d '{"match": {"nw_dst": "{ip
tujuan}", "nw_proto": "{protokol}", "tp_dst":
{port}, "actions":{"queue": {nomor}}'
http://localhost:8080/qos/rules/{switch}
```

4.3. Implementasi Klaster Hadoop

Sub-bab ini membahas mengenai implementasi klaster Hadoop yang dibangun. Klaster Hadoop dibangun diatas arsitektur SDN. Terdapat enam tahapan untuk mengimplementasikan klaster Hadoop.

4.3.1. Menginstal Java dan Hadoop

Tahapan pertama adalah melakukan instalasi Java dikarenakan Hadoop menggunkan bahasa pemograman Java. Java belum terinstal pada Raspberry Pi. Setelah Java berhasil terpasang langkah berikutnya adalah melakukan konfigurasi ssh antara Datanode dan juga Namenode. Tahapan berikutnya adalah melakukan instalasi Hadoop. Hadoop dapat diunduh dihalam web resminya. Pada Tugas Akhir ini Hadoop yang digunakan Hadoop versi 1.2.

4.3.2. Konfigurasi *Master*

Langkah kedua adalah melakukan konfigurasi *master*. Konfigurasi *master* dilakukan pada file *master*. File *master* berisi tentang *hostname* dari *master* atau Namenode. Fungsi dari file ini adalah agar Datanode mengetahui *master* atau Namenode-nya. Saat pertama kali mendapatkan Hadoop, file ini tidak ada didalam direktori manapun Hadoop. File *master* dapat dibuat dengan perintah sebagai berikut.

```
% sudo nano $HADOOP_HOME/conf/master
```

4.3.3. Konfigurasi Slave

Langka ketiga adalah melakukan konfigurasi *slave*. File *slave* berisi tentang *hostname* dari node node yang akan dijadikan *slaves* atau Datanode. Fungsi dari file ini adalah agar Namenode mengetahui *slave* atau Datanode yang terhubung. File untuk melakukan konfigurasi *slave* dapat dilakukan dengan perintah sebagai berikut:

```
% sudo nano $HADOOP_HOME/conf/slaves
```

4.3.4. Konfigurasi Hadoop Env dan Core

Tahapan ke-enam adalah melakukan konfigurasi *core* Hadoop. *Hadoop-env.sh* berisi tentang konfigurasi lingkungan Hadoop saat berjalan. Konfigurasi lingkungan dapat berupa Java yang dipakai oleh Hadoop saat mengeksekusi program, dan ukuran *heap*. Untuk melakukan konfigurasi *hadoop-env.sh* dapat dilakukan dengan perintah sebagai berikut.

```
sudo nano $HADOOP_HOME/conf/hadoop-env.sh
```

File *core-site.xml* berisi pengaturan konfigurasi penting (*core*) Hadoop. Pengaturan alamat Namenode, nomor port yang digunakan Hadoop, alokasi memori untuk sistem, batas memori penyimpanan data, dan ukuran ukuran *buffer* untuk proses *read/write*. Untuk melakukan perubahan file core-site.xml dapat dilakukan dengan perintah sebagai berikut.

```
sudo nano $HADOOP_HOME/conf/core-site.xml
```

Pada *core-site.xml* dilakukan pengaturan *hadoop.tmp.dir* dan juga *fs.default.name*. *Hadoop.tmp.dir* adalah memberikan letak spesifik dari direktori tempat penyimpanan dari HDFS dan penyimpanan data file sistem dari MapReduce. Ketika melakukan format, maka isi dari folder ini perlu dihapus secara manual. Sedangkan *fs.default.name* memberikan secara manual nama *hostname* dari Namenode dan port yang digunakan.

4.3.5. Konfigurasi MapRed

Tahapan berikutnya adalah melakukan konfigurasi MapRed. Untuk melakukan konfigurasi MapRed dilakukan pada file *mapred-site.xml*. Untuk pertama kali penggunaan *mapred-site.xml* dapat didapatkan templatenya dari *$HADOOP_HOME/conf/mapred-site.xml.template*. File *mapred-site.xml* dapat diakses dengan perintah sebagai berikut.

```
% sudo nano $HADOOP_HOME/conf/mapred-site.xml
```

Konfigurasi *mapred-site.xml* untuk Namenode dan Datanode berbeda. Untuk Namenode konfigurasi yang paling penting adalah memberi nilai pada kelas *mapred.job.tracker*. Nilai tersebut adalah nama *hostname* Namenode dan port yang akan dialokasikan. Untuk Datanode konfigurasi nilai pada kelas *mapred.job.tracker* harus sama seperti nilai *mapred.job.tracker* pada Namenode.

4.3.6. Konfigurasi HDFS

Tahapan berikutnya adalah melakukan konfigurasi HDFS. Untuk melakukan konfigurasi HDFS dilakukan pada file *hdfs-site.xml*. HDFS menangani seluruh data yang disimpan pada Hadoop. Pada konfigurasi HDFS dilakukan pengaturan jumlah replikasi data, jalur Namenode, dan lokasi Datanode pada komputer tiap *slave*. Untuk melakukan perubahan konfigurasi mengenai HDFS dapat dilakukan dengan menjalankan perintah sebagai berikut.

```
% sudo nano $HADOOP_HOME/conf/hdfs-site.xml
```

Pada konfigurasi HDFS dilakukan pengaturan nilai *dfs.replication. Dfs.replication* mengatur total replikasi data yang disimpan ke dalam HDFS. Selain itu juga terdapat *dfs.block.size* mengatur nilai blok dari data yang akan disimpan pada HDFS. Pada klaster Hadoop yang dibangun diberi nilai dari *dfs.replication*

1 menyebabkan data yang disimpan kedalam hadoop hanya memiliki 1 data.

Pada implementasi Tugas Akhir ini ukuran *block size* diubah tidak menjadi 64 MB. Pengaturan ukuran *block data* di inisialisasikan pada variabel *dfs.block size*. Memperkecil blok data dari HDFS mempertimbangkan memori Raspberry yang relatif rendah, ukuran file yang kecil.

Setiap perubahan yang dilakukan pada *hdfs-site.xml* harus memformat ulang namenode dan menghapus semua direktori *(hadoop.tmp.dir*). Hal ini disebabkan Namenode selalu mencatat *metadata* setelah melakukan penyimpanan. Peletakan data pada Datanode yang sudah diatur oleh Namenode, akan kacau jika data tidak di-format. Hal ini dapat berakibat hilang atau tidak terdefinisinya blok data dalam Hadoop.

4.4. Hadoop dan SDN

Pada sub-bab ini akan dijelaskan implementasi klaster Hadoop pada arsitektur SDN yang dibangun, terdapat dua jenis aplikasi yang akan digunakan yaitu aplikasi switch dan router.

4.4.1. Switch

Pada sub-bab ini kan dijelaskan implementasi pengaturan *transfer rate* untuk klaster Hadoop yang dibangun. Keluaran dari proses ini membuat klaster Hadoop yang memeiliki pengaturan lalu lintas jaringan sehingga proses perpindahan data pada Hadoop dapat optimal dan tidak terganggu dengan *congestion*.

Agar dapat dilakukan pengaturan *transfer rate* maka perlu dijalankan aplikasi SDN untuk melakukan pengaturan *transfer rate* selain penghubung antara switch dan juga host. Menjalankan aplikasi SDN dilakukan dengan menjalankan perintah:

```
% ryu-manager ryu.app.rest_qos
ryu.app.qos_simple_switch_13
ryu.app.rest_conf_switch
```

Setelah mengatur datanode dan juga namenode pada proses implementasi sebelumnya langkah berikutnya adalah membuat *queue*. *Queue* yang akan dibuat sejumlah dua *queue*. *Queue* pertama untuk Hadoop dan *queue* kedua untuk lalu lintas yang dilakukan oleh Iperf. Pengaturan *queue* dilakukan melalui kontroler SDN. Inisialisasi *queue* dilakukan pada tiap port yang tersambung. Membuat *queue* dapat dilakukan dengan cara menjalankan perintah:

```
% curl -X POST -d '{"port_name": "{namaport}",
"type": "linux-htb", "max_rate": {nilai maksimal},
"queues": [{pengaturan queue 1},  queuedua}]}
'http://localhost:8080/qos/queue/switch
```

Queue yang dibuat berjumlah dua. *Queue* pertama untuk lalu lintas jaringan Hadoop dan program pendukung Hadoop, *queue* kedua dibuat untuk lalu lintas Iperf. Pada tiap *queue* dapat diatur nilai maksimum dan juga minimum *transfer rate*. *Queue* untuk Hadoop memiliki nilai lebih besar dari *queue* untuk Iperf. Tahap berikutnya memetakan *queue* ke spesifik *host*. Memetakan *queue* ke host dapat dilakukan dengan menjalankan perintah:

```
% curl -X POST -d '{"match": {"nw_dst": "{ip
tujuan",
"nw_proto":"Protokol","tp_dst":"5002"},"actions":
{"queue":{queue }}'http://localhost:8080/qos/rule
s/{switch}
```

4.4.2. Dua Switch dengan Aplikasi Router

Pada sub-bab ini akan dijelaskan implementasi klaster Hadoop pada arsitektur jaringan SDN menggunkan dua switch.

```
% sed '/OFPFlowMod(/,/)/s/0, cmd/1, cmd/'
ryu/ryu/app/rest_router.py >
ryu/ryu/app/qos_rest_router.py
```

Agar dua switch dapat saling terhubung digunakan aplikasi SDN router. Kedua Switch dihubungkan menggunakan kabel LAN. Terdapat sedikit modifikasi kode program pada aplikasi router Ryu. Jalankan perintah berikut untuk memasuka *flow entry* ke tabel id: 1.

File *qos_rest_router* yang dibentuk dilakukan modifikasi. Modifikasi dilakukan pada bagian fungsi *packetin_to_node*. Pada Aplikasi Ryu diberikan batasan panjang *buffer* apabila paket yang masuk melebihi nilai panjang *buffer* yang diatur paket akan di *drop*. Agar dapat mengirim paket yan melebihi panjang *buffer* yang ditentukan dilakukan modifikasi program yang dapat dilihat pada

```
1  def _packetin_to_node(self, msg,header_list):
2    #Pengecekan panjang buffer dihilangkan
3    in_port=self.ofctl.get_packetin_inport(msg)
4    src_ip = None
5    dst_ip = header_list[IPV4].dst
6    srcip = ip_addr_ntoa(header_list[IPV4].src)
7    dstip = ip_addr_ntoa(dst_ip)
```

Setelah dilakukan perubahan, instal kembali aplikasi yang telah dibuat dengan menjalankan perintah:

```
% cd ryu/; python ./setup.py install
```

Untuk menjalankan aplikasi SDN router yang telah dilakukan modifikasi dapat dijalankan dengan perintah:

```
% ryu-manager ryu.app.rest_qos
ryu.app.qos_rest_router ryu.app.rest_conf_switch
```

Ketika aplikasi telah berjalan dilakukan pengaturan seperti yang telah dijelaskan pada sub-bab aplikasi router. Seperti memberikan nilai IP pada tiap antarmuka pengaturan *routing* dan juga pengaturan *transfer rate.*

BAB V
PENGUJIAN DAN EVALUASI

Bab ini membahas pengujian dan evaluasi pada klaster Hadoop yang dibangun. Pengujian yang dilakukan adalah pengujian pengujian arsitektur SDN, dan klaster Hadoop. Pengujian pengaturan manajemen *transfer rate* juga dilakukan untuk mengetahui pengaruh dari manajemen *transfer rate*. Hasil evaluasi menjabarkan tentang hasil pengujian pada bagian akhir bab ini.

5.1. Lingkungan Pengujian

Lingkungan pengujian sistem pada pengerjaan Tugas Akhir ini dilakukan pada lingkungan dan alat kakas seperti yang tertera pada Tabel 5.1.

Tabel 5.1 Lingkungan Pengujian Sistem

Switch	Perangkat Keras: -Raspberry Pi 3 model B Sistem operasi: -Raspbian Jesse *lite* Prosesor : - 1.2GHz 64-bit quad-core ARMv8 CPU Memori : - 1 GB
Kontroler	Perangkat Keras: - Laptop Sistem operasi: -Ubuntu 14.04 Prosesor : - Intel® Core™ 2 Duo P870 CPU2.53 GHz Memori : - 2 GB Perangkat Lunak: -Ryu

Host Jenis 1	Perangkat Keras: -Raspberry Pi 2 model B Sistem operasi: -Raspbian Jesse *lite* Prosesor : - 900 MHz quad-core ARM Cortex-A7 CPU ARMv7 Memori : - 1GB
Host Jenis 2	Perangkat Keras: -Raspberry Pi model B Sistem operasi: -Raspbian Jesse *lite* Prosesor : 700 MHz ARM1176JZF-S Core CPU ARMv6 Memori : - 512MB

5.2. Pengujian Arsitektur SDN

Pengujian arsitektur SDN dilakukan untuk mengetahui apakah arsitektur SDN yang dibangun dapat berjalan dan ketiga komponen dapat bekerja sesuai dengan hasil yang diharapkan. Pada pengujian arsitektur SDN akan dicoba aplikasi yang telah ada pada arsitektur yang telah dibuat.

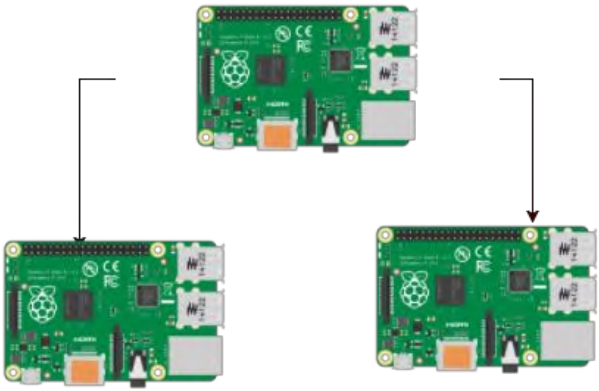
5.2.1. Skenario Pengujian Aplikasi Switch

Pada subbab ini dijelaskan beberapa skenario uji coba aplikasi SDN switch pada arsitektur yang dibangun. Terdapat tiga skenario uji coba yang dilakukan, pada tiap uji coba akan dilakukan evaluasi. Ketiga skenario tersebuat adalah:

1. Satu switch
2. Dua switch dihubungkan dengan GRE-*tunnel*
3. Dua switch dihubungkan dengan kabel LAN

5.2.1.1. Satu Switch

Pengujian satu switch dilakukan dengan cara menjalankan aplikasi SDN *simple switch* pada arsitektur yang dibangun dengan menggunakan satu switch, seperti yang dapat dilihat pada Gambar 5.1. Pada uji coba ini juga dilakukan uji coba mendapatkan jumlah host maksimal yang dapat terhubung dan juga ukuran *throughput* yang didapatkan. Pengukuran *throughput* dilakukan dengan menjalankan program Iperf.



Gambar 5.1 Satu Switch

Hasil uji coba menjalankan aplikasi *simple switch* sekaligus skenario uji coba dapat dilihat pada Tabel 5.2.

Tabel 5.2 Uji Coba Satu Switch Aplikasi *Simple switch*

Nama	Uji coba aplikasi switch hub dengan 1 switch
Tujuan Uji Coba	Menguji fungsionalitas aplikasi switch dan infrastruktur yang dibangun
Kondisi Awal	Aplikasi dijalankan
Skenario	1. Menghubungkan host ke switch 2. Antar host melakukan ping dan juga ssh
Masukan	-
Keluaran	Antar host dapat saling terhubung.
Hasil Uji Coba	Berhasil

Pada skenario satu switch juga dilakukan uji coba mendapatkan jumlah host yang dapat terhubung pada switch. Hasil uji coba dapat dilihat pada Tabel 5.3

Tabel 5.3 Uji Coba Jumlah Host

Jumlah host yang terhubung	Hasil
4	Switch mati
3	Berhasil
2	Berhasil
1	Berhasil

Dilakukan juga uji coba mendapatkan nilai maksimum *transfer rate*. Nilai dari uji coba didapatkan dari menjalankan program Iperf. Satu host menjalankan *server* dan host satunya menjadi klien. Hasil uji coba dapat dilihat pada Tabel 5.4.

Tabel 5.4 Uji Coba Pengukuran *Throughput*

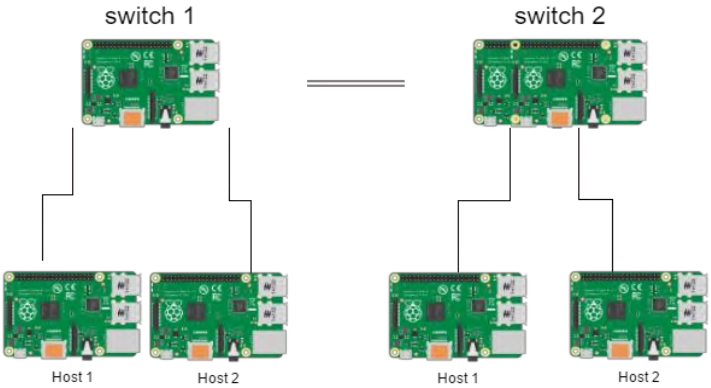
Protokol	*Throughput*(Mbits/detik)
TCP	93
UDP	93

5.2.1.2. Evaluasi Satu Switch

Berdasarkan hasil uji coba yang dilakukan aplikasi switch dapat berjalan pada model satu switch. Jumlah host yang dapat terhubung dengan switch didapatkan nilai maksimum tiga host. Penyebab tidak dapat sebanyak empat host dikarenakan permasalahan *power supply* switch dikarenakan pada saat menyambungkan empat host switch Raspberry Pi mati. Pada manual Raspberry Pi di tulis semakin banyak antarmuka yang terhubung maka kebutuhan listrik Raspberry Pi bertambah. Pada model satu switch nilai *throughput* sebesar 93 Mbit/detik. Tes hanya dilakukan pada satu *traffic flow* saja. Nilai *throughput* dapat menurun apabila lalu lintas jaringan pada *platform* SDN yang dibangun bertambah.

5.2.1.3. Dua Switch GRE-*Tunnel*

Pada skenario dua switch ini dilakukan dengan cara menjalankan aplikasi SDN *simple switch* pada arsitektur yang dibangun, kedua switch dihubungkan dengan cara membuat GRE-*tunnel*, seperti yang dapat dilihat pada Gambar 5.2. Pada uji coba ini juga dilakukan uji coba nilai *throughput* yang didapatkan. Pengukuran *throughput* dilakukan dengan menjalankan program Iperf.



Gambar 5.2 Dua Switch dengan GRE-*tunnel*

Hasil uji coba menjalankan aplikasi *simple switch* sekaligus skenario uji coba dapat dilihat pada Tabel 5.5.

Tabel 5.5 Hasil Uji Coba Dua Switch dengan GRE-*Tunnel*

Nama	Uji coba aplikasi switch hub dua switch Gre tunnel
Tujuan Uji Coba	Menguji fungsionalitas aplikasi switch dan infrastruktur yang dibangun
Kondisi Awal	Aplikasi dijalankan
Skenario	1. Menghubungkan host ke switch 2. Antar host melakukan ping dan juga ssh
Masukan	-
Keluaran	Antar host dapat saling terhubung.
Hasil Uji Coba	Berhasil

Pada skenario ini pada tahap awal antar host sempat tidak bisa melakukan ssh namun dapat melakukan ping. Oleh karena itu dilakukan uji coba juga mencari besar MTU yang dapat dilihat pada Tabel 5.6 sehingga antar host dapat saling melakukan ssh.

Tabel 5.6 Uji Coba Besar MTU

Ukuran MTU	Hasil
1500	SSH gagal
1400	Berhasil

Dilakukan juga uji coba mendapatkan nilai *throughput*. Nilai dari uji coba didapatkan dari menjalankan program Iperf. Satu host menjalankan *server* dan host satunya menjadi klien. Hasil uji coba dapat dilihat pada Tabel 5.7

Tabel 5.7 Pengukuran *Throughput*

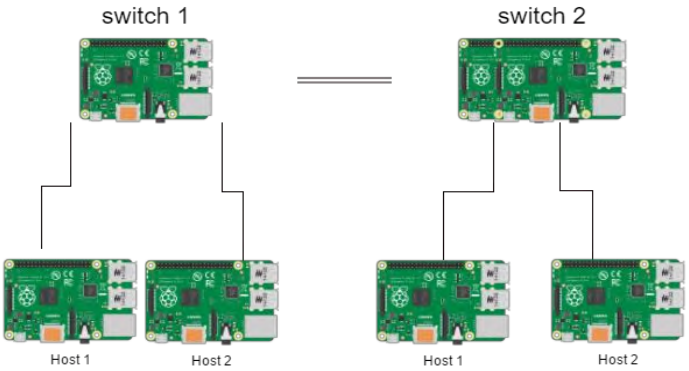
Protokol	*Throughput* (Mbit/detik)
TCP	3,6
UDP	3.6

5.2.1.3.1. Evaluasi Dua Switch dengan GRE-*Tunnel*

Berdasarkan hasil uji coba yang dilakukan aplikasi switch dapat berjalan pada model dua switch dengan menggunkan GRE-*tunnel*. Pembuatan GRE-*tunnel* menggunkan fitur yang dimiliki oleh switch OpenFlow. Pada model ini besaran MTU berpengaruh. Hal itu dapat dilihat dari percobaan pada Tabel 5.6. Pada percobaan pertama kedua host dapat terhubung namun tidak dapat saling melakukan ssh. Pada percobaan kedua nilai besaran MTU diubah menjadi 1400, kedua host dapat saling melakukan ssh. Pada model dua switch dengan GRE-*tunnel* didapatkan nilai *throughput* sebesar 3,6 Mbit/detik. Tes hanya dilakukan pada satu *traffic flow* saja. Nilai *throughput* dapat menurun apabila lalu lintas jaringan pada *platform* SDN yang dibangun bertambah. Dari hasil yang didapatkan terjadi penurunan nilai yang cukup signifikan anatara model satu switch dengan model dua switch ini.

5.2.1.4. Dua Switch dengan Kabel LAN

Pada skenario dua switch ini dilakukan dengan cara menjalankan aplikasi SDN *simple switch* pada arsitektur yang dibangun, kedua switch dihubungkan dengan cara membuat GRE-*tunnel*, seperti yang dapat dilihat pada Gambar 5.3. Pada uji coba ini juga dilakukan uji coba nilai *throughput* yang didapatkan. Pengukuran *throughput* dilakukan dengan menjalankan program Iperf.



Gambar 5.3 Uji Coba GRE-*tunnel*

Hasil uji coba menjalankan aplikasi *simple switch* sekaligus skenario uji coba dapat dilihat pada Tabel 5.8.

Tabel 5.8 Hasil Uji Coba Dua Switch dengan Kabel LAN

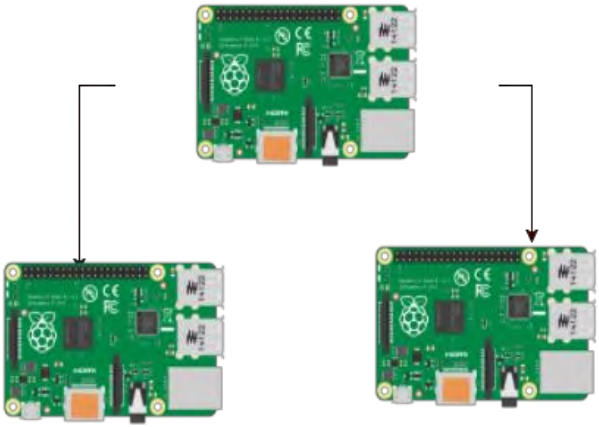
Nama	Uji coba aplikasi switch hub dua switch dengan kabel LAN
Tujuan Uji Coba	Menguji fungsionalitas aplikasi switch dan infrastruktur yang dibangun
Kondisi Awal	Aplikasi dijalankan
Skenario	1. Menghubungkan host ke switch 2. Antar host melakukan ping dan juga ssh
Masukan	-
Keluaran	Antar host dapat saling terhubung.
Hasil Uji Coba	Berhasil

5.2.1.5. Evaluasi Dua Switch dengan Kabel LAN

Pada uji coba didapatkan alamat fisik yang sama pada antarmuka jaringan, mengakibatkan kedua switch mendapatkan id yang sama sehingga tidak dapat tersambung. Permasalahan tersebut dapat terselesaikan dengan mengganti alamat fisik antarmuka jaringan adapter USB *to* ethernet yang digunakan atau mengganti id switch OpenFlow.

5.2.2. Pengaturan *Transfer Rate*

Pada subbab ini dijelaskan skenario uji coba aplikasi SDN pengaturan *transfer rate* pada arsitektur yang dibangun. Pada uji coba aplikasi pengaturan *transfer rate* menggunkan model satu switch yang dapat dilihat pada Gambar 5.4.



Gambar 5.4 Uji Coba Pengaturan *Transfer Rate*

Hasil uji coba menjalankan pengaturan *transfer rate* sekaligus skenario uji coba dapat dilihat pada Tabel 5.9. Pada saat menjalankan pengaturan *transfer rate* juga dijalankan aplikasi switch. Pengaturan *transfer rate* pada aplikasi ini dilakukan dengan cara membuat *queue*. *Queue* akan dipetakan ke port dengan IP dan juga protokol tujuan.

Tabel 5.9 Uji Coba Pengaturan *Transfer Rate*

Nama	Uji Coba aplikasi pengaturan *transfer rate* dengan satu switch
Tujuan Uji Coba	Menguji fungsionalitas aplikasi pengaturan *queue* dan pengaturan infrstruktur yang dibangun
Kondisi Awal	Aplikasi dijalankan
Skenario	1. Membuat *queue* 2. Mengatur *flow entry*, dengan cara memasukan IP, port tujuan dengan *queue* yang telah dibuat
Masukan	Aturan tiap *queue*, IP tujuan , port tujuan, protokol , dan juga aturan *queue*
Keluaran	*Transfer rate* berubah sesuai dengan pengaturan
Hasil Uji Coba	Berhasil

Dilakukan juga uji coba untuk mendapatkan pengaturan kecepatan *transfer* maksimum yang akan digunakan untuk nilai maksimal pada *queue* yang digunakan untuk proses manajemen *transfer rate* pada proses HDFS Hadoop. Hasil uji coba dapat dilihat pada Tabel 5.10.

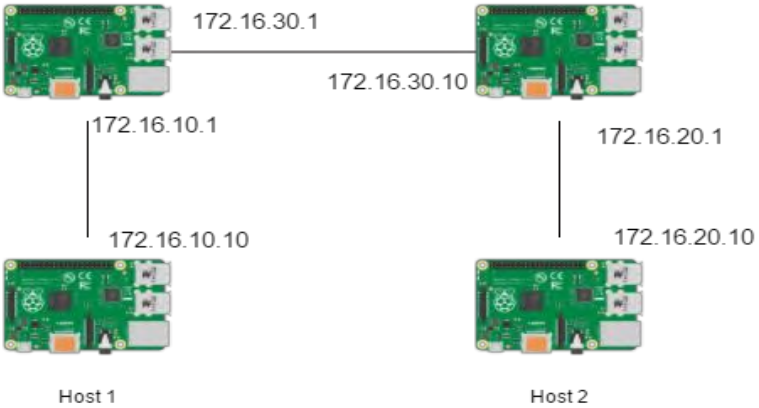
Tabel 5.10 Hasil Uji Coba Kecepatan *Transfer* Maksimum

Kecepatan transfer maksimum (Mbit/detik)	Hasil
50	Gagal- switch *kernel panic*
25	Gagal- switch *kernel panic*
12	Gagal- switch *kernel panic*
5	Gagal- switch *kernel panic*
1	Berhasil

Dari hasil uji coba didapatkan pengaturan nilai *transfer* maksimum hanya sebesar 1Mbits/detik, apabila mendapatkan yang lebih besar switch mengalami *kernel panic.*

5.2.3. Router

Pada sub-bab ini dijelaskan skenario uji coba aplikasi SDN router pada arsitektur yang dibangun. Pada uji coba aplikasi router menggunkan dua switch dengan topologi linear, dua switch dihubungkan secara serial, seperti yang dapat dilihat pada Gambar 5.5.



Gambar 5.5 Topologi uji Coba Router

Hasil uji coba menjalankan aplikasi *simple switch* sekaligus skenario uji coba dapat dilihat pada Tabel 5.11.

Tabel 5.11 Hasil Uji Coba Aplikasi Router

Nama	Uji coba aplikasi Router
Tujuan Uji Coba	Menguji fungsionalitas *routing*
Kondisi Awal	Aplikasi Routing telah dijalankan
Skenario	1. Memasukan alamat interface yang akan dibentuk untuk switch 1 dan 2 2. Memasukan *default gateway* untuk switch 1 dan 2
Masukan	Alamat IP tiap antarmuka, dan IP *default gateway*
Keluaran	Terbentuk antarmuka dan rute
Hasil Uji Coba	Berhasil melakukan *routing*

Dilakukan juga uji coba mendapatkan nilai *throughput*. Nilai dari uji coba didapatkan dari menjalankan program Iperf. Satu host menjalankan *server* dan host satunya menjadi klien. Hasil uji coba dapat dilihat pada Tabel 5.12.

Tabel 5.12 Pengujian *Throughput*

Protokol	*Throughput* (Mbit/detik)
TCP	1,6
UDP	1.6

Pada uji coba menggunakan aplikasi Router juga dilakukan uji coba mengirimkan data dengan SCP pada Tabel 5.13.

Tabel 5.13 Uji Coba Perpindahan Data dengan SCP

Nama	Uji coba pengiriman data aplikasi Router
Tujuan Uji Coba	Menguji pengiriman data pada arsitektur dengan aplikasi router
Kondisi Awal	Aplikasi router telah dijalankan
Skenario	Mengirimkan data dengan menggunakan SCP dari host satu ke host dua
Masukan	File yang akan dipindahkan
Keluaran	File berhasil dipindahkan
Hasil Uji Coba	Berhasil memindahkan data

5.3. Pengujian Hadoop

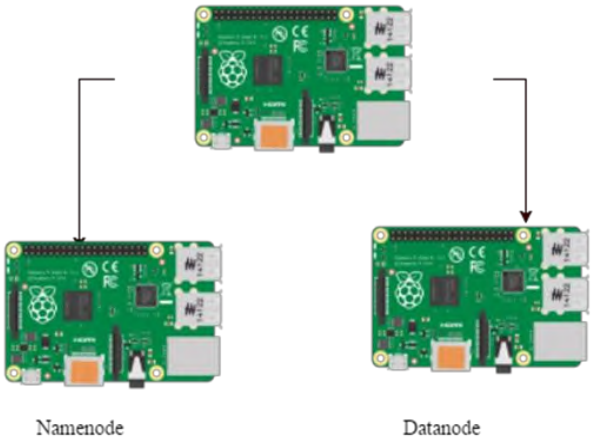
Pengujian klaster Hadoop dilakukan untuk mengetahui apakah klaster Hadoop dapat berjalan pada arsitektur SDN yang dibangun. Pada pengujian Hadoop dicoba dijalankan proses MapReduce dan juga HDFS. Terdapat beberapa skenario uji coba menjalankan Hadoop pada arsitektur yang dibangun. Terdapat dua skenario uji coba yang dilakukan, pada tiap uji coba akan dilakukan evaluasi. Kedua skenario tersebuat adalah:

1. Satu switch

2. Dua switch
3. Tanpa menggunkan SDN

5.3.1. Satu Switch

Pada subbab ini dijelaskan uji coba Hadoop pada model satu switch. Pengujian dilakukan dengan cara menjalankan program MapReduce dan juga proses penyimpana data ke HDFS. Klaster Hadoop pada model switch dapat dilihat pada Gambar 5.6 dimana terdiri dari satu Namenode dan juga satu Datanode. Memasukan data ke HDFS akan dilakukan dari Namenode. Pada proses ini dijalankan menggunkan aplikasi SDN switch.



Gambar 5.6 Klaster Hadoop Satu Switch

Hasil uji coba dan skenario uji coba proses penyimpanan data ke HDFS dapat dilihat pada Tabel 5.14, sementara hasil uji coba dan juga skenario uji coba MapReduce dapat dilihat pada Tabel 5.15. Pada proses data masukan dimulai dari 1MB, 10MB, 20MB.

Tabel 5.14 Hasil Uji Coba Penyimpan HDFS

Nama	Menjalankan penyimpanan data ke HDFS
Tujuan Uji Coba	Menguji proses HDFS pada klaster Hadoop
Kondisi Awal	Data belum dimasukan kedalam HDFS datanode
Skenario	Memulai Hadoop, kemudian melakukan penyimpanan kedalam HDFS
Masukan	Memasukan data berupa teks
Keluaran	Data dimasukan kedalam HDFS yang berada pada data node
Hasil Uji Coba	Berhasil.
Keberhasilan	8/10

Tabel 5.15 Hasil Uji Coba MapReduce

Nama	Menjalankan MapReduce
Tujuan Uji Coba	Menguji kluster Hadoop dan menjalankan MapReduce
Kondisi Awal	Data tersimpan dalam datanode
Skenario	1. Memulai Hadoop 2. Menjalankan program MapReduce
Masukan	Program *wordcount*
Keluaran	Hasil perhitungan *word* count
Hasil Uji Coba	Berhasil.
Keberhasilan	7/10

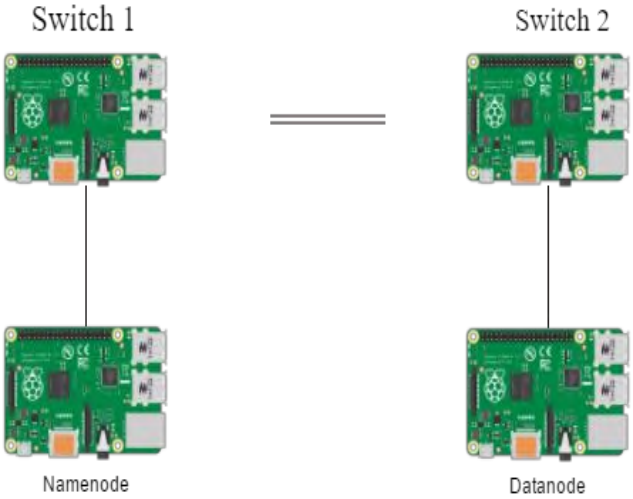
5.3.2. Evaluasi Hadoop Satu Switch

Pada saat menjalankan HDFS dan juga MapReduce terkadang switch mengalami *kernel panic* namun intensitasnya tidak tinggi. Dari sepuluh kali menjalankan program MapReduce didapatkan kegagalan karena *kernel panic* sejumlah tiga kali.

Sedangkan Pada proses HDFS sebanyak dua kali mengalamai kegagalan disebabkan *kernel panic*.

5.3.3. Dua Switch

Pada subbab ini dijelaskan uji coba Hadoop pada model dua switch. Pengujian dilakukan dengan cara menjalankan program MapReduce dan juga proses penyimpana data ke HDFS. Klaster Hadoop pada model switch dapat dilihat pada Gambar 5.7 dimana terdiri dari dua switch. Pada switch pertama terdapat Namenode dan pada switch kedua tersambung Datanode dari satu Namenode dan juga satu Datanode. Memasukan data ke HDFS akan dilakukan dari Namenode.



Gambar 5.7 Klaster Hadoop dengan Dua Switch

Hasil uji coba dan skenario uji coba proses penyimpanan data ke HDFS dapat dilihat pada Tabel 5.16, sementara hasil uji coba dan juga skenario uji coba MapReduce dapat dilihat pada Tabel 5.17.

Tabel 5.16 Hasil Uji Coba HDFS Dua Switch

Nama	Menjalankan penyimpanan data ke HDFS
Tujuan Uji Coba	Menguji kluster Hadoop
Kondisi Awal	Data belum dimasukan kedalam HDFS datanode
Skenario	1. Memulai Hadoop 2. Melakukan penyimpanan kedalam HDFS
Masukan	Memasukan data berupa teks
Keluaran	Data dimasukan kedalam HDFS yang berada pada data node
Hasil Uji Coba	Berhasil
Keberhasilan	6/10

Tabel 5.17 Hasil Uji Coba MapReduce Dua Switch

Nama	Menjalankan MapReduce
Tujuan Uji Coba	Menguji kluster Hadoop dan menjalankan MapReduce
Kondisi Awal	Data tersimpan dalam datanode
Skenario	1. Memulai Hadoop 2. Menyimpan data ke HDFS 3. Menjalankan program MapReduce
Masukan	Program *wordcount*
Keluaran	Hasil perhitungan *word* count
Hasil Uji Coba	Berhasil.
Keberhasilan	4/10

5.3.4. Evaluasi Klaster Hadoop Dua Switch

Pada saat menjalankan HDFS dan juga MapReduce switch mengalami *kernel panic*. Dari sepuluh kali menjalankan program MapReduce didapatkan kegagalan karena *kernel panic* sejumlah tujuh kali. Sedangkan Pada proses HDFS sebanyak empat kali mengalamai kegagalan disebabkan *kernel panic*.

5.3.5. Hadoop Tanpa Aplikasi SDN

Pada subbab ini dijelaskan perbandingan uji coba Hadoop tanpa menggunakan aplikasi SDN dan menggunakan aplikasi SDN pada switch. Pengujian dilakukan dengan cara menjalankan proses penyimpanan data ke HDFS. Klaster Hadoop pada model switch dapat dilihat pada Gambar 5.6 dimana terdiri dari satu Namenode dan juga satu Datanode. Memasukan data ke HDFS akan dilakukan dari Namenode. Pada uji coba ini terdapat dua skenario. Skenario pertama yang dapat dilihat pada Tabel 5.18 dilakukan penyimpanan data dengan menggunakan aplikasi SDN. Skenario kedua yang dapat dilihat pada Tabel 5.20 dilakuan penyimpanan data tanpa menggunakan aplikasi SDN. Uji coba ini bertujuan untuk mendapatkan nilai perbandingan proses penyimpanan data ke HDFS antara menggunakan SDN dan tanpa SDN.

Tabel 5.18 Uji Coba Aplikasi SDN

Nama	Menjalankan penyimpanan data ke HDFS
Tujuan Uji Coba	Mendapatkan waktu proses HDFS pada klaster Hadoop dengan SDN
Kondisi Awal	1. Data belum dimasukan kedalam HDFS Datanode. 2. Menjalakan aplikasi SDN.
Skenario	Memulai Hadoop, kemudian melakukan penyimpanan kedalam HDFS.
Masukan	Memasukan data berupa teks
Keluaran	Data dimasukan kedalam HDFS yang berada pada data node
Hasil Uji Coba	Berhasil.

Pada proses data masukan dimulai dari 1MB, 10MB, 20MB. Dari skenario tersebut didapatkan hasil uji coba yang dapat dilihat pada Tabel 5.19.

Tabel 5.19 Waktu Proses HDFS SDN

Ukuran	Waktu (Menit:Detik)				
1 MB	34.8	33.6	33.4	30.1	30.4

10 MB	5:43	5:22	5:23	5:09	5:21
20 MB	10:55	11:29	11:00	11:02	11:03

Tabel 5.20 Uji Coba Aplikasi Tanpa SDN

Nama	Menjalankan penyimpanan data ke HDFS
Tujuan Uji Coba	Mendapatkan waktu proses HDFS pada klaster Hadoop tanpa SDN
Kondisi Awal	1. Data belum dimasukan kedalam HDFS Datanode. 2. Tidak menjalakan aplikasi SDN.
Skenario	Memulai Hadoop, kemudian melakukan penyimpanan kedalam HDFS.
Masukan	Memasukan data berupa teks
Keluaran	Data dimasukan kedalam HDFS yang berada pada data node
Hasil Uji Coba	Berhasil.

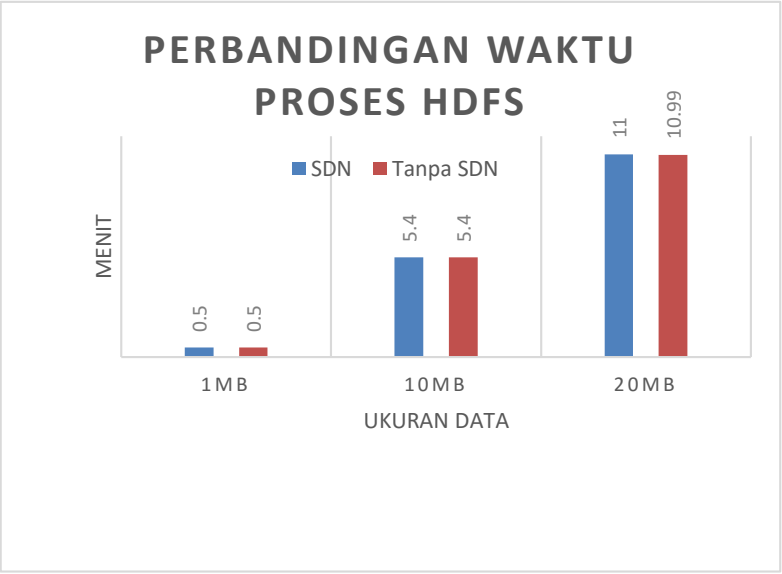
Pada proses data masukan dimulai dari 1MB, 10MB, 20MB. Dari skenario tersebut didapatkan hasil uji coba yang dapat dilihat pada Tabel 5.21.

Tabel 5.21 Waktu Proses HDFS Tanpa SDN

Ukuran	Waktu (Menit:Detik)				
1 MB	34.8	34.7	33.3	32.1	32.1
10 MB	5:21	5:08	5:13	5:42	5:21
20 MB	10:51	11:29	11:25	10:55	11:27

5.3.6. Evaluasi Tanpa Aplikasi SDN

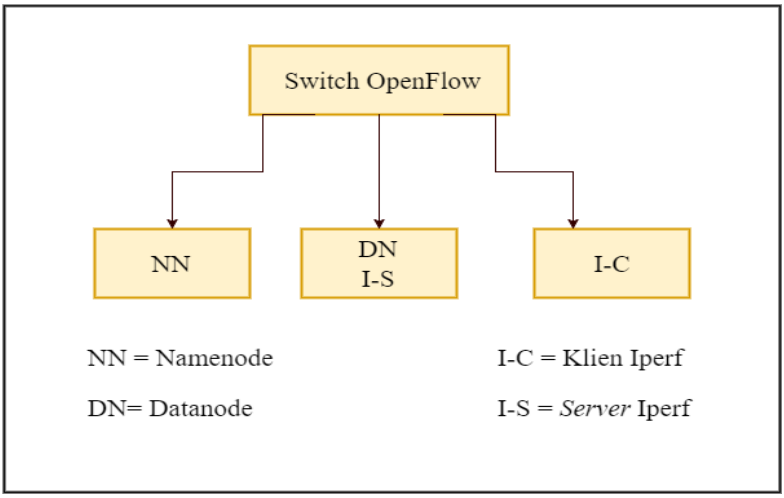
Pada saat menjalankan proses penyimpanan data ke HDFS tanpa menggunkan aplikasi SDN membutuhkan waktu yang relatif sama dengan menggunakan aplikasi SDN pada switch. Perbandinagn waktu proses HDFS dengan SDN dan tanpa menggunkan SDN dapat dilihat pada Gambar 5.8



Gambar 5.8 Grafik Perbandingan Waktu Proses HDFS

5.4. Manajemen *Transfer rate* pada Proses Hadoop HDFS

Pengujian manajemen *transfer rate* pada proses Hadoop dilakukan untuk mengetahui apakah manajemen *transfer* rate dapat berpengaruh pada proses HDFS pada klaster Hadoop yang dibangun. Pada pengujian ini dicoba dijalankan proses penyimpanan data ke HDFS. Pada manajemen *transfer rate* digunakan model satu switch dikarenakan memiliki tingkat keberhasilan paling tinggi dibandingkan model lain seperti yang dapat dilihat pada Gambar 5.9. Pada switch akan tersambung tiga host. Host pertama merupakan Namenode, host kedua Datanodeyang bertindak juga sebagai *server* Iperf, dan host ketiga menjadi klien Iperf. Pada uji coba ini terdapat tiga skenario uji coba.



Gambar 5.9 Topologi Pengujian Manajemen *Transfer Rate*

5.4.1. Skenario Pertama

Skenario pertama proses HDFS Hadoop berjalan tanpa ada lalu lintas tambahan dari program Iperf, nilai dari skenario pertama dijadikan tolak ukur waktu proses HDFS. Skenario uji coba dapat dilihat pada Tabel 5.22.

Tabel 5.22 Uji Coba *Transfer Rate* Skenario Pertama

Nama	Skenario pertama manajemen *transfer rate*
Tujuan Uji Coba	Mendapatkan nilai tolak ukur waktu pada proses HDFS Hadoop
Kondisi Awal	Data belum tersimpan kedalam HDFS
Skenario	Dari Namenode melakukan penyimpanan data ke HDFS yang berada pada Datanode
Masukan	Data teks
Keluaran	Data teks tersimpan pada HDFS
Hasil Uji Coba	Berhasil

Pada skenario pertama dilakukan memasukan data ke HDFS dengan tiga jenis ukuran yaitu 1MB, 10MB, dan 20MB. Hasil dari tiap proses dapat dilihat pada Tabel 5.23.

Tabel 5.23 Waktu Proses HDFS Skenario Pertama

Ukuran Data	Waktu (Menit:Detik)
1MB	33.4
10MB	5:22
20MB	11:00

5.4.2. Skenario Kedua

Skenario kedua pada saat menjalankan proses HDFS Hadoop diberikan lalu lintas tambahan yang berjalan pada *queue* yang sama. Pemberian lalu lintas tambahan dengan cara klien Iperf mengirimkan data berupa TCP ke Datanode sehingga lalu lintas data proses penyimpanan ke HDFS terjadi *congestion*. Tujuan dari skenario ini untuk mendapatkan waktu apabila pada proses HDFS terdapat lalu lintas tambahan yan mengakibatkan *congestion*. Skenario dan hasil uji coba dapat dilihat pada Tabel 5.24.

Tabel 5.24 Skenario Kedua Manajemen *Transfer Rate*

Nama	Skenario kedua manajemen *transfer rate*
Tujuan Uji Coba	Mendapatkan nilai waktu pada proses HDFS Hadoop apabila terdapat *congestion* tanpa dilakukan manajemen *transfer rate*
Kondisi Awal	Data belum tersimpan kedalam HDFS
Skenario	1. Dari Namenode melakukan penyimpanan data ke HDFS yang berada pada Datanode 2. Program Iperf dijalankan, klien Iperf mengirim paket kepada *server* Iperf (Datanode) sehingga lalu lintas pada Datanode bertambah
Masukan	Data teks
Keluaran	Data teks tersimpan pada HDFS

Dari skenario tersebut didapatkan hasil uji coba yang dapat dilihat pada Tabel 5.25. Waktu proses HDFS mengalami peningkatan.

Tabel 5.25 Waktu Proses HDFS Skenario Kedua

Ukuran	Waktu (Menit:Detik)				
1 MB	52.897	55.879	51.81	52.204	55.384
10 MB	7:4	7:26	7:17	7:17	7:25
20 MB	13:11	13:30	13:25	13:27	13:12

5.4.3. Skenario Ketiga

Skenario ketiga sama seperti skenario kedua namun dilakukan manajemen *transfer rate*. Lalu lintas data program Iperf dipisahkan dengan lalu lintas penyimpanan data ke HDFS. Lalu lintas program Iperf dimasukan kedalam q_1 dan lalu lintas data HDFS dimasukan ke q_0. Q_0 diberikan nilai *transfer rate* yang lebih besar dari nilai *transfer rate* q_1. Pengaturan *queue* dapat dilihat pada Tabel 5.26. Skenario uji coba dapat dilihat pada Tabel 5.27.

Tabel 5.26 Pengaturan *Queue*

Queue	Nilai maksimum	Nilai Minimum
Q_0	1Mbps	800 Kbps
Q_1	300 Kbps	-

Tabel 5.27 Skenario Ketiga Manajemen *Transfer Rate*.

Nama	Skenario kedua manajemen *transfer rate*
Tujuan Uji Coba	Mendapatkan nilai waktu pada proses HDFS Hadoop apabila terdapat *congestion* tanpa dilakukan manajemen *transfer rate*
Kondisi Awal	Data belum tersimpan kedalam HDFS
Skenario	1. Dari Namenode melakukan penyimpanan data ke HDFS yang berada pada Datanode 2. Program Iperf dijalankan, klien Iperf mengirim paket kepada *server* Iperf

	(Datanode) sehingga lalu lintas pada Datanode bertambah 3. Lalu lintas Iperf masuk kedalam q_1 4. Lalu lintas HDFS Hadoop masuk kedalam q_0.
Masukan	Data teks
Keluaran	Data teks tersimpan pada HDFS

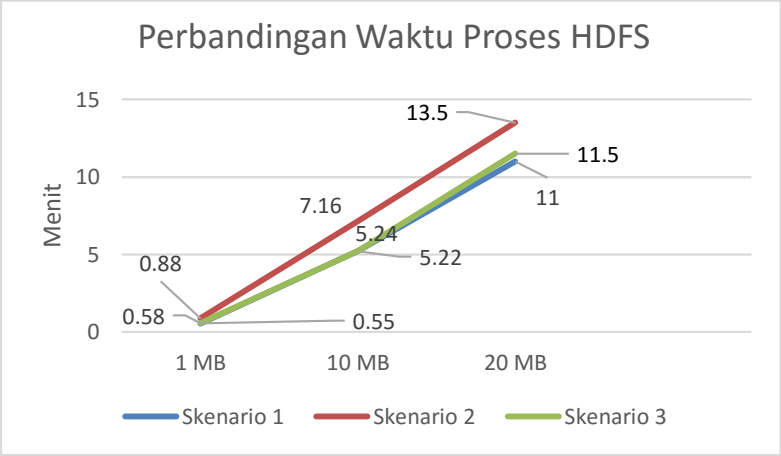
Dari skenario tersebut dilakukan didapatkan hasil uji coba yang dapat dilihat pada Tabel 5.28.

Tabel 5.28 Hasil Uji Coba Skenario Tiga

Ukuran	Waktu (Menit:Detik)				
1 MB	35.810	35.353	35.373	36.655	34.213
10 MB	5:44	5:30	5:25	5 :25	5:33
20 MB	11:23	11:23	11:52	10:54	11:34

5.4.4. Hasil Uji Coba Manajemen *Transfer Rate*

Berdasarkan hasil uji coba ketiga skenario yang dapat dilihat pada Gambar 5.10. Proses HDFS mengalami peningkatan waktu apabila terdapat lalu lintas tambahan pada klaster Hadoop, hal tersebut dapat dilihat dari perbandingan skenario satu dan dua. Melakukan manajemen *transfer rate* pada klaster Hadoop dapat menurukan waktu proses HDFS hal tersebut dapat dilihat dari perbandingan skenario dua dan tiga. Dengan melakukan manajemen *transfer rate* waktu proses HDFS Hadoop menjadi tidak terlalu banyak peningkatan walaupun dalam klaster Hadoop terdapat lalu lintas tambahan yang dapat mengakibatkan *congestion*.



Gambar 5.10 Grafik Perbandingan Waktu HDFS

[Halaman ini sengaja dikosongkan]

LAMPIRAN

Tabel A.1 Uji Coba Aplikasi Switch

No	Hasil	Host/switch	Analisis
1	Gagal	4	Permasalahan *supply* power, untuk percobaan berikutnya digunakan supply power yang berbeda
2	Gagal	4	Permasalahan *supply power*, *host* yang tersambung dikurangkan
3	Berhasil	3	-
4	Berhasil	3	-
5	Berhasil	3	-
6	Berhasil	3	-
7	Berhasil	3	-
8	Berhasil	3	-
9	Berhasil	3	-
10	berhasil	3	-

Tabel A.2 Uji Coba Aplikasi Switch dengan GRE-Tunnel

No	Hasil	Host/switch	Analisis
1	Gagal	2	Ukuran MTU
2	Berhasil	2	Menganti ukuran MTU menjadi 1400
3	Berhasil	2	-
4	Berhasil	2	-
5	Berhasil	1	-
6	Berhasil	1	-
7	Berhasil	1	-
8	Berhasil	1	-
9	Berhasil	1	-
10	berhasil	1	-

Tabel A.3 Uji Coba Dua Switch dengan Kabel LAN

No	Status	Host/switch	Analisis
1	Gagal	Sw 1 =1 Sw2 =2	ID switch sama
2	Berhasil	Sw 1 =1 Sw2 =2	Mengubah alamat fisik antarmuka jaringan adapter USB port
3	Berhasil	Sw 1 =1 Sw2 =2	-
4	Berhasil	Sw 1 =1 Sw2 =2	-
5	Berhasil	Sw 1 =1 Sw2 =2	-
6	Berhasil	Sw 1 =1 Sw2 =2	-
7	Berhasil	Sw 1 =1 Sw2 =2	-
8	Berhasil	Sw 1 =1 Sw2 =2	-
9	Berhasil	Sw 1 =1 Sw2 =2	-
10	berhasil	Sw 1 =1 Sw2 =2	-

Tabel A.4 Uji Coba Aplikasi Router

No	Status	Host/switch	Analisis
1	Gagal	1	ID switch sama
2	Berhasil	2	Mengubah alamat fisik antarmuka jaringan adapter USB port
3	Berhasil	1	Setelah 10 menit baru dapat tersambuing anatar host
4	Berhasil	1	Menurunkan nilai variable mac learning pada kode

5	Berhasil	1	-
6	Berhasil	1	-
7	Berhasil	1	Terputus setelah didiamkan beberapa lama
8	Berhasil	1	-
9	Berhasil	1	Terputus setelah didiamkan beberapa lama mengelurakan notifikasi *"switch leaves"*
10	berhasil	1	-

Gambar A.1 Rak Switch Rapsberry Pi

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini akan diberikan kesimpulan yang diambil selama pengerjaan Tugas Akhir serta saran-saran tentang pengembangan yang dapat dilakukan terhadap Tugas Akhir ini di masa yang akan datang.

6.1. Kesimpulan

Dari hasil selama proses perancangan, implementasi, serta pengujian dapat diambil kesimpulan sebagai berikut:

1. Melakukan manajemen *transfer rate* dapat mengoptimalkan proses HDFS Hadoop pada klaster Hadoop.
2. Dengan menetapkan nilai transfer rate pada klaster Hadoop maka lalu lintas untuk proses penyimpanan data ke HDFS terbebas dari *congestion* yang disebabkan dari lalu lintas data lain.
3. Manajemen *transfer rate* dapat dilakukan dengan memanfaatkan fitur *queue* yang dimiliki switch OpenFlow.
4. Arsitektur SDN dapat dibangun dengan memanfaatkan Raspberry Pi sebagai switch OpenFlow.
5. Permasalahan besar *bandwith* dan *bottleneck* pada switch menjadi kendala pada saat menjalankan proses MapReduce yang membutuhkan *bandwith* yang besar dan juga proses yang banyak.
6. Permasalahan *power supply* dapat mempengaruhi kinerja Raspberry Pi.
7. Aplikasi SDN tidak mempengaruhi waktu proses penyimpanan data ke HDFS.

6.2. Saran

Berikut saran-saran untuk pengembangan dan perbaikan sistem di masa yang akan datang. Diantaranya adalah sebagai berikut:

1. Melakukan pengaturan *transfer rate* dapat dilakukan secara dinamis.
2. Mekanisme konfigurasi perangkat memakan waktu yang lama dan berulang dapat dilakukan dengan menggunakan *script.*

DAFTAR PUSTAKA

[1] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," 2014.

[2] N. Mckeown, T. Anderson, H. Balakrishman, G Parulkar, L. Peterson, J. Rexford, S. Shenker J.turner, "Openflow: Enabling Innovation In Campus Networks," 2008.

[3] S. Narayan, S. Bailey dan A. Daga, "Hadoop acceleration in an OpenFlow-Based cluster".

[4] H. Kim, J. Kim and Y.-B. Ko, "Developing Cost-effective OpenFlow Test Bed for Small Scale Software Defined networking," 2014.

[5] Open networking foundation, "SDN Architecture," 2014.

[6] T. White, Hadoop The Definitive Guide, California: O'REILLY, 2009.

[7] "Pantou : OpenFlow 1.0 for OpenWRT," [Online]. Available: http://archive.openflow.org/wk/index.php/Pantou[Accessed 10 Juni 2016].

[8] "Open vSwitch," [Online]. Available: http://openvswitch.org/. [Accessed 10 Juni 2016].

[9] "CPqD OpenFlow Switch," [Online]. Available: https://github.com/CPqD/ofsoftswitch13.. [Accessed 10 Juni 2016].

[10] "Mininet," [Online]. Available: http://mininet.org/. [Accessed 10 Juni 2016].

[11] "Net-fpga," [Online]. Available: http://netfpga.org/. [Accessed 10 Juni 2016].

[12] "POX," [Online]. Available: ttps://openflow.stanford.edu/display/ONL/POX. [Accessed 10 Juni 2016].

[13] "Beacon," [Online]. Available: openflow.stanford.edu/display/Beacon/Home. [Accessed 10 Juni 2016].

[14] "OpenDayLight," [Online]. Available: https://www.opendaylight.org/.

[15] "Ryu," [Online]. Available: http://github.com/osrg/ryu. [Accessed 10 Juni 2016].

[16] "OVSDB," [Online]. Available: https://tools.ietf.org/html/draft-pfaff-ovsdb-proto-00. [Accessed 10 Juni 2016].

[17] "Raspberry Pi," [Online]. Available: www.raspberrypi.org. [Accessed 10 Juni 2016].

BIODATA PENULIS

Narendra Hanif Wicaksana atau yang biasa dipanggil Hanif, lahir di Jakarta, 22 April 1994. Memiliki 2 saudara kandung.

Penulis menempuh pendidikan di SD Al-Falah (1999-2006), SMP Labschool Jakarta (2006-2009), dan SMA Labschool Jakarta (2009-2012). Setelah lulus SMA penulis melanjutkan ke jenjang perkuliahan di Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember Surabaya. Bidang Studi yang diambil oleh penulis pada saat kuliah di Teknik Informatika ITS adalah Komputasi Berbasi Jaringan.

Selama menempuh kuliah penulis mengikuti kegiatan pengembangan kepribadian baik dari institut, jurusan dan luar institut. Penulis pernah menjadi asisten dosen dan praktikum untuk mata kuliah Dasar Pemograman (2015). Selama menempuh kuliah penulis pernah memenangi beberapa lomba seperti, Compfest. Penulis pernah menjadi finalis dari lomba Intel® RealSense™ Challenge pada Compfest 2014, dan Juara Hackathon Merderka 2.0 penaggulangan kabut asap. Penulis dapat dihubungi melalui alamat *email* hanifnarendra@gmail.com.