



TUGAS AKHIR - KI141502

**Implementasi *Wireless Quality of Service*
dengan Metode *Load Switching* Jaringan Seluler
Menggunakan *Software Defined Network* untuk
Meningkatkan *Network Reliability* pada
Jaringan Dinamis**

**YOGA BAYU AJI PRANAWA
5113100023**

**Dosen Pembimbing
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.
Waskitho Wibisono, S.Kom., M.Eng., Ph.D.**

**JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017**

[Halaman ini sengaja dikosongkan]



TUGAS AKHIR - KI141502

**Implementasi *Wireless Quality of Service*
dengan Metode *Load Switching* Jaringan Seluler
Menggunakan *Software Defined Network* untuk
Meningkatkan *Network Reliability* pada
Jaringan Dinamis**

**YOGA BAYU AJI PRANAWA
5113100023**

**Dosen Pembimbing
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.
Waskitho Wibisono, S.Kom., M.Eng., Ph.D.**

**JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017**

[Halaman ini sengaja dikosongkan]



TUGAS AKHIR - KI141502

**Implementation of Wireless Quality of Service
by Load Switching Method for Cellular Network
using Software Defined Network to Improve
Network Reliability in Dynamic Network**

**YOGA BAYU AJI PRANAWA
5113100023**

Advisor

**Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.
Waskitho Wibisono, S.Kom., M.Eng., Ph.D.**

**DEPARTMENT OF INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2017**

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

IMPLEMENTASI *WIRELESS QUALITY OF SERVICE* DENGAN METODE *LOAD SWITCHING* JARINGAN SELULER MENGGUNAKAN *SOFTWARE DEFINED* *NETWORK* UNTUK MENINGKATKAN *NETWORK* *RELIABILITY* PADA JARINGAN DINAMIS

TUGAS AKHIR

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

Oleh:

YOGA BAYU AJI PRANAWA

NRP: 5113 100 023

Disetujui oleh Dosen Pembimbing Tugas Akhir

Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.
NIP: 197708242006041001

(pembimbing 1)

Waskitho Wibisono, S.Kom., M.Eng., Ph.D.
NIP: 197410222000031001

(pembimbing 2)

SURABAYA,
JANUARI 2016



[Halaman ini sengaja dikosongkan]

**IMPLEMENTASI *WIRELESS QUALITY OF SERVICE*
DENGAN METODE *LOAD SWITCHING* JARINGAN
SELULER MENGGUNAKAN *SOFTWARE DEFINED
NETWORK* UNTUK MENINGKATKAN *NETWORK
RELIABILITY* PADA JARINGAN DINAMIS**

Nama Mahasiswa : Yoga Bayu Aji Pranawa
NRP : 5113100023
Jurusan : Teknik Informatika - FTIf ITS
Dosen Pembimbing 1 : Royyana Muslim Ijtihadie, S.Kom.,
M.Kom., Ph.D.
Dosen Pembimbing 2 : Waskitho Wibisono, S.Kom.,
M.Eng., Ph.D.

ABSTRAK

Wireless Quality of Service (QoS) adalah salah satu dimensi mobilitas, yaitu sebuah metode yang digunakan untuk menjaga kualitas suatu jaringan nirkabel. *QoS* diperlukan sebagai sebuah metode untuk memenuhi kriteria pelayanan sistem bagi pengguna, yaitu *confidentiality*, *integrity*, dan *availability*. Beberapa aspek yang menjadi topik utama dalam *QoS* adalah *failure and recovery mechanism*, *variable bandwidth*, *computing distribution*, *discovery mechanism*, *variable latency*, dan *performance feedback*. *Wireless* yang dibahas pada Tugas Akhir ini dititik beratkan pada jaringan seluler yang cenderung tidak *reliable* pada daerah tertentu. Oleh karena itu dibutuhkan sebuah mekanisme yang dapat mengatasi tidak stabilnya jaringan seluler tersebut.

Dalam Tugas Akhir ini dibangun suatu mekanisme untuk menerapkan *wireless quality of service*. Implementasi mekanisme yang diterapkan pada Tugas Akhir ini adalah dengan menerapkan *load switching* pada jaringan seluler dengan menggunakan beberapa provider. Provider yang dipilih adalah

provider yang menggunakan frekuensi GSM dengan mempertimbangkan beberapa aspek, diantaranya adalah network reachable, round trip time, dan throughput. Beberapa aspek tersebut kemudian dilakukan komputasi sehingga menghasilkan sebuah parameter yang dijadikan pedoman pemilihan jaringan.

Implementasi Tugas Akhir ini menggunakan load switching pada Software Defined Network (SDN) dengan protokol OpenFlow. Software Defined Network ini digunakan sebagai framework komunikasi pada suatu arsitektur jaringan. Aplikasi load switching yang digunakan akan menentukan sebuah flow berdasarkan parameter yang telah dibuat.

Uji coba yang diterapkan pada Tugas Akhir ini meliputi uji coba fungsionalitas dan uji coba performa dengan menggunakan beberapa kondisi skenario yang telah ditentukan. Berdasarkan hasil uji coba dapat disimpulkan bahwa sistem yang dibuat pada Tugas Akhir ini dapat menerapkan wireless quality of service dan meningkatkan network reliability sebesar 65,29% dan 83,87% lebih baik untuk penggunaan tanpa waktu tunggu dan dengan waktu tunggu pada suatu jaringan dinamis.

Kata kunci: wireless quality of service, software defined network, load switching, jaringan seluler, OpenFlow.

IMPLEMENTATION OF WIRELESS QUALITY OF SERVICE BY LOAD SWITCHING METHOD FOR CELLULAR NETWORK USING SOFTWARE DEFINED NETWORK TO IMPROVE NETWORK RELIABILITY IN DYNAMIC NETWORK

Name : Yoga Bayu Aji Pranawa
NRP : 5113100023
Department : Teknik Informatika - FTIf ITS
Supervisor 1 : Royyana Muslim Ijtihadie, S.Kom.,
M.Kom., Ph.D.
Supervisor 2 : Waskitho Wibisono, S.Kom.,
M.Eng., Ph.D.

ABSTRACT

Wireless Quality of Service (QOS) is the one of mobility dimension, that is a method to keep the quality of the wireless network. QOS is required as a method to meet the criteria of system services to client, that is confidentiality, integrity, and availability. Some aspects of QOS main topic is failure and recovery mechanism, variable bandwidth, computing distribution, discovery mechanism, variable latency, and performance feedback. Wireless in this final project refers to cellular network that tend to be unreliable in some areas. Therefore, it needs some mechanism to handle the stability of the cellular network.

This final project has been developed some mechanism to implementing wireless quality of service. Implementation of that mechanism is applying a load switching algorithm in cellular network by using multiple provider. Provider has been chosen by using GSM frequency and considered by some aspects, including network reachable, round trip time, and throughput. Some of that aspects calculated and produce some parameter to select the best network later.

Implementation of this final project is using load switching in Software Defined Network (SDN) with OpenFlow protocol. Software Defined Network is used as communication framework in some network architectures. load switching application used to consider flow according to the calculated parameter.

Testing implementation of this final project includes functionality test and performance test with determined scenarios. According to the test result, it can be concluded that system in this final project can provide wireless quality of services and improve network reliability 65,29% and 83,87% better without waiting time and using waiting time in the dynamic network.

Kata kunci: wireless quality of service, software defined network, load switching, cellular network, OpenFlow.

KATA PENGANTAR



Alhamdulillahirabbil'alamin, segala puji bagi Allah Subhannahu Wata'alla, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul ***“Implementasi Wireless Quality of Service dengan Metode Load Switching Jaringan Seluler untuk Meningkatkan Network Reliability pada Jaringan Dinamis”*** dengan baik.

Proses pembuatan Tugas Akhir ini tidak lepas dari berbagai pihak yang telah ikut berperan sehingga Tugas Akhir ini dapat diselesaikan dengan baik. Penulis ingin mengungkapkan rasa terimakasih sebesar-besarnya kepada:

1. Allah SWT atas limpahan rahmat dan hidayah kepada penulis sehingga penulis dapat menyelesaikan Tugas Akhir ini dengan baik.
2. Bapak Sunardi, dan Ibu Nur Achiriyati selaku Bapak dan Ibu penulis yang selalu memberikan dukungan moral maupun material serta motivasi untuk penulis, dan selalu mengingatkan kepada penulis untuk tidak pantang menyerah dalam menghadapi masalah.
3. Bapak Royyana Muslim Ijtihadie dan Bapak Waskitho Wibisono selaku dosen pembimbing yang telah memberikan bimbingan, arahan, kritik, saran, bantuan, dan dukungan untuk menyelesaikan Tugas Akhir ini.
4. Bapak Imam Kuswardayan selaku dosen wali yang telah memberikan nasihat, teguran, dan motivasi kepada penulis dalam menjalani perkuliahan di Teknik Informatika.
5. Bapak Baskoro Adi Pratomo dan Bapak Hudan Studiawan selaku dosen rumpun mata kuliah jaringan yang telah

memberikan saran dan masukan teknis pada pengerjaan Tugas Akhir ini.

6. Bapak/Ibu dosen dan segenap *staff* karyawan Jurusan Teknik Informatika FTIF-ITS yang telah mendidik penulis selama menjalani kuliah di Jurusan Teknik Informatika.
7. Sahabat Keluarga Mahasiswa Klaten di Surabaya yang telah memberikan semangat dan dukungan kepada penulis sehingga penulis dapat menyelesaikan Tugas Akhir.
8. Kak Thiar Hasbia, Kak Surya Dharma Putra, M. Syaiful Jihad A, Daniel Fablius, I G N Adi Wicaksana, Setiyo Adi W, Nindyasari Dewi U dan seluruh teman-teman laboratorium Arsitektur Jaringan Komputer yang ikut membantu memberikan ide dan solusi pengerjaan Tugas Akhir ini.
9. Saudara angkatan 2013 yang telah menemani dan memotivasi sejak awal perkuliahan pada tahun pertama sampai saat ini.
10. Seluruh kakak-kakak dan teman-teman di Kementrian Kominfo BEM ITS Kolaborasi serta pengurus BEM ITS Kolaborasi yang telah memberikan kesempatan untuk belajar dan mencari pengalaman berorganisasi.
11. Sahabat “*God Bless You*” yang telah menghibur, memberikan semangat dan memberikan motivasi sehingga penulis dapat menyelesaikan Tugas Akhir.
12. Dan seluruh kerabat, teman, dan saudara yang tidak dapat disebutkan satu per satu.

Buku ini tentu jauh dari kata sempurna. Oleh karena itu, penulis memohon maaf atas segala kesalahan dan kekurangan. Saran dan kritik yang membangun sangat penulis harapkan dari pembaca.

Surabaya, 20 Januari 2017

Yoga Bayu Aji Pranawa

DAFTAR ISI

LEMBAR PENGESAHAN.....	vii
KATA PENGANTAR.....	xiii
DAFTAR ISI.....	xv
DAFTAR GAMBAR.....	xix
DAFTAR TABEL.....	xxiii
DAFTAR KODE SUMBER.....	xxv
BAB I PENDAHULUAN.....	1
1.1. Latar Belakang.....	1
1.2. Rumusan Permasalahan.....	3
1.3. Batasan Masalah.....	3
1.4. Tujuan.....	4
1.5. Manfaat.....	4
1.6. Metodologi.....	4
1.7. Sistematika Penulisan Tugas Akhir.....	6
BAB II TINJAUAN PUSTAKA.....	9
2.1. Wireless Quality of Service.....	9
2.2. Load Switching.....	9
2.3. Jaringan Seluler.....	12
2.4. Software Defined Network.....	13
2.4.1. <i>Controller</i>	14
2.4.2. OpenFlow.....	15
2.4.3. REST (<i>Representational State Transfer</i>) API..	16
2.5. Dynamic Network Area.....	17
2.6. Wireless Local Area Network.....	18
BAB III ANALISIS DAN PERANCANGAN SISTEM.....	21
3.1. Deskripsi Umum Sistem.....	21
3.2. Arsitektur Sistem.....	23
3.3. Perancangan Arsitektur Jaringan.....	25
3.4. Diagram Alir Aplikasi.....	27
3.4.1. Diagram Alir Inisialisasi Sistem.....	27
3.4.2. Diagram Alir <i>Controller</i>	28

3.4.3. Diagram Alir <i>Host Router</i>	29
3.4.4. Diagram Alir <i>Load Switching</i>	30
3.4.5. Diagram Alir Pengambilan Parameter.....	32
3.4.6. Diagram Alir Penentuan <i>Flow</i>	32
3.4.7. Diagram Alir Pengambilan Statistik <i>Throughput Controller</i>	34
3.5. Perancangan Antarmuka <i>Logging Sistem</i>	34
BAB IV IMPLEMENTASI	37
4.1. Lingkungan Implementasi.....	37
4.1.1. Lingkungan Implementasi Perangkat Keras.....	37
4.1.2. Lingkungan Implementasi Perangkat Lunak.....	38
4.2. Implementasi Perangkat Lunak.....	39
4.2.1. Implementasi pada <i>Controller</i>	39
4.2.1.1. Instalasi <i>Controller</i>	39
4.2.1.2. Proses Inisialisasi <i>Controller</i>	40
4.2.1.3. Konfigurasi <i>Controller</i>	40
4.2.1.4. Proses Inisialisasi <i>load switching</i>	41
4.2.1.5. Proses Pengambilan Statistik <i>Throughput</i>	42
4.2.1.6. Proses Memasukkan Data ke <i>Database</i>	42
4.2.1.7. Proses Penentuan <i>Flow</i>	43
4.2.2. Implementasi pada <i>Router</i>	44
4.2.2.1. Instalasi Fitur OpenFlow.....	44
4.2.2.2. Konfigurasi OpenFlow.....	44
4.2.3. Implementasi <i>Host Router</i>	46
4.2.3.1. Proses <i>Dial-Up</i>	46
4.2.3.2. Proses <i>Forwarding</i>	47
4.2.3.3. Proses Pengambilan dan Pengiriman <i>Throughput</i>	47
4.3. Implementasi Rute.....	48
4.3.1. Analisis <i>Coverage</i>	49
4.3.2. Analisis <i>Download</i>	51
4.3.3. Analisis <i>Upload</i>	54

4.4. Implementasi Antarmuka Perangkat Lunak.....	56
BAB V UJI COBA DAN EVALUASI.....	59
5.1. Lingkungan Uji Coba.....	59
5.1.1. Perangkat Keras.....	59
5.1.2. Perangkat Lunak.....	61
5.1.3. Rute Perjalanan.....	62
5.2. Uji Coba Fungsionalitas.....	63
5.2.1. Inisialisasi <i>Controller</i>	64
5.2.2. Inisialisasi <i>Load Switching</i>	66
5.2.3. Pengambilan Statistik <i>Throughput</i>	68
5.2.4. Memasukkan Data ke <i>Database</i>	70
5.2.5. Penentuan <i>Flow</i>	71
5.2.6. Proses <i>Dial-Up</i>	74
5.2.7. <i>Forwarding</i> Paket.....	76
5.2.8. Pengambilan dan Pengiriman <i>Throughput</i>	77
5.2.9. Inisialisasi <i>Network Monitoring</i>	80
5.3. Uji Coba Performa.....	82
5.3.1. Uji Performa <i>Network Reachable</i>	83
5.3.1.1. Uji Performa <i>Network Reachable</i> tanpa <i>Load Switching</i>	83
5.3.1.2. Uji Performa <i>Network Reachable</i> dengan <i>Load Switching</i> tanpa Waktu Tunggu.....	85
5.3.1.3. Uji Performa <i>Network Reachable</i> dengan <i>Load Switching</i> menggunakan Waktu Tunggu....	86
5.3.2. Uji Performa <i>Round Trip Time</i>	88
5.3.2.1. Uji Performa <i>Round Trip Time</i> tanpa <i>Load</i> <i>Switching</i>	88
5.3.2.2. Uji Performa <i>Round Trip Time</i> dengan <i>Load Switching</i> tanpa Waktu Tunggu.....	89
5.3.2.3. Uji Performa <i>Round Trip Time</i> dengan <i>Load Switching</i> menggunakan Waktu Tunggu....	91
5.3.3. Uji Performa Pengunduhan.....	93

5.3.3.1. Uji Performa Pengunduhan tanpa <i>Load Switching</i>	93
5.3.3.2. Uji Performa Pengunduhan dengan <i>Load Switching</i> tanpa Waktu Tunggu.....	94
5.3.3.3. Uji Performa Pengunduhan dengan <i>Load Switching</i> menggunakan Waktu Tunggu.....	96
5.3.4. Uji Performa Pengunggahan.....	97
5.3.4.1. Uji Performa Pengunggahan tanpa <i>Load Switching</i>	98
5.3.4.2. Uji Performa Pengunggahan dengan <i>Load Switching</i> tanpa Waktu Tunggu.....	99
5.3.4.3. Uji Performa Pengunggahan dengan <i>load switching</i> menggunakan Waktu Tunggu.....	100
5.3.5. Ringkasan Uji Performa.....	102
5.3.5.1. Penghitungan <i>Delay Time</i>	102
5.3.5.2. Penghitungan Tingkat <i>Reliability</i>	103
BAB VI PENUTUP.....	105
6.1. Kesimpulan.....	105
6.2. Saran.....	106
DAFTAR PUSTAKA.....	107
LAMPIRAN.....	109
BIODATA PENULIS.....	125

DAFTAR GAMBAR

Gambar 2.1. Algoritma Round Robin	10
Gambar 2.2. Algoritma <i>Least Connection</i>	11
Gambar 2.3. Algoritma <i>Hash</i>	12
Gambar 2.4. Geometri Jaringan Seluler	13
Gambar 2.5. Arsitektur <i>Controller</i> OpenDaylight Helium	15
Gambar 2.6. Fitur OpenFlow	16
Gambar 2.7. Konfigurasi <i>Wireless Local Area Network</i>	19
Gambar 2.8. Perbandingan <i>Mobile Data Network</i> menggunakan Grafik Kivat	19
Gambar 3.1. Arsitektur Sistem secara Umum.....	23
Gambar 3.2. Arsitektur Jaringan OpenFlow	23
Gambar 3.3. Proses Penanganan Paket pada OpenFlow	24
Gambar 3.4. Diagram Cara Kerja REST API	25
Gambar 3.5. Diagram Sistem <i>Host Router</i>	25
Gambar 3.6. Perancangan Arsitektur Jaringan.....	27
Gambar 3.7. Diagram Alir Inisialisasi Sistem.....	28
Gambar 3.8. Diagram Alir <i>Controller</i>	29
Gambar 3.9. Diagram Alir <i>Host Router</i>	30
Gambar 3.10. Diagram Alir <i>Load Balancer</i>	31
Gambar 3.11. Diagram Alir Pengambilan Parameter.....	32
Gambar 3.12. Diagram Alir Penentuan <i>Flow</i>	33
Gambar 3.13. Diagram Alir Pengambilan Statistik <i>Throughput Controller</i>	34
Gambar 3.14. Tampilan Antarmuka <i>Logging</i> Sistem.....	35
Gambar 4.1. Proses Inisialisasi <i>Controller</i>	40
Gambar 4.2. Perintah Pengaturan <i>Environment</i>	40
Gambar 4.3. Perintah Menjalankan Aplikasi <i>Controller</i>	40
Gambar 4.4. Alamat Antarmuka <i>Controller</i>	41
Gambar 4.5. Perintah Menjalankan Inisialisasi <i>load switching</i> . 41	
Gambar 4.6. <i>Pseudocode</i> Inisialisasi <i>load switching</i>	42
Gambar 4.7. <i>Pseudocode</i> Pengambilan Statistik <i>Throughput</i>	42

Gambar 4.8. <i>Pseudocode</i> Memasukkan Data ke Database.....	43
Gambar 4.9. <i>Pseudocode</i> Penentuan <i>Flow</i>	43
Gambar 4.10. Antarmuka <i>Router</i> dengan Protokol OpenFlow..	44
Gambar 4.11. Konfigurasi OpenFlow pada <i>Router</i>	46
Gambar 4.12. Proses Instalasi Aplikasi <i>Dial-up</i>	47
Gambar 4.13. Konfigurasi Proses <i>Forwarding Host Router</i>	47
Gambar 4.14. <i>Pseudocode</i> Proses Pengambilan dan Pengiriman <i>Throughput</i>	48
Gambar 4.15. <i>Coverage Area</i> menggunakan <i>Provider</i> Telkomsel.....	49
Gambar 4.16. <i>Coverage Area</i> menggunakan <i>Provider XL</i>	50
Gambar 4.17. <i>Coverage Area</i> menggunakan <i>Provider Indosat</i> Ooredoo.....	50
Gambar 4.18. <i>Coverage Area</i> menggunakan <i>Provider Three</i>	51
Gambar 4.19. Representasi Kecepatan Unduh <i>Provider</i> Telkomsel.....	52
Gambar 4.20. Representasi Kecepatan Unduh <i>Provider XL</i>	52
Gambar 4.21. Representasi Kecepatan Unduh <i>Provider Indosat</i> Ooredoo.....	53
Gambar 4.22. Representasi Kecepatan Unduh <i>Provider Three</i> .	53
Gambar 4.23. Representasi Kecepatan Unggah <i>Provider</i> Telkomsel.....	54
Gambar 4.24. Representasi Kecepatan Unggah <i>Provider XL</i> ...	55
Gambar 4.25. Representasi Kecepatan Unggah <i>Provider Indosat</i> Ooredoo.....	55
Gambar 4.26. Representasi Kecepatan Unggah <i>Provider</i> Three.....	56
Gambar 4.27. Tampilan Antarmuka <i>Monitoring Grafana</i>	57
Gambar 5.1. Implementasi Uji Coba pada <i>Dashboard Mobil</i> .	61
Gambar 5.2. Rute yang Ditempuh untuk Uji Coba.....	63
Gambar 5.3. Perintah untuk menjalankan <i>Controller</i>	65
Gambar 5.4. Tampilan <i>Log</i> Inisialisasi <i>Controller</i>	65
Gambar 5.5. Tampilan Antarmuka <i>Controller</i>	66

Gambar 5.6. Perintah Inisialisasi <i>load switching</i>	67
Gambar 5.7. Tampilan Antarmuka <i>load switching</i>	68
Gambar 5.8. Tampilan Antarmuka Proses Pengambilan Statistik <i>Throughput</i>	69
Gambar 5.9. Antarmuka Proses Memasukkan Data <i>Throughput</i> ke <i>Database</i>	71
Gambar 5.10. Tampilan Penentuan Master Port <i>Flow</i> pada <i>Terminal Linux</i>	73
Gambar 5.11. Tampilan Perpindahan <i>Flow</i> pada <i>Terminal Linux</i>	73
Gambar 5.12. Tampilan Penambahan <i>Flow</i> pada <i>Browser</i>	74
Gambar 5.13. Tampilan Antarmuka Proses <i>Dial-Up</i>	75
Gambar 5.14. Tampilan Antarmuka Uji Coba <i>Forwarding Paket</i>	77
Gambar 5.15. Tampilan Antarmuka Proses Pengambilan dan Pengiriman <i>Throughput</i> pada <i>Terminal Linux</i>	79
Gambar 5.16. Tampilan Antarmuka Perubahan <i>Rx Bytes</i> pada <i>Browser</i>	80
Gambar 5.17. Perintah Melakukan Inisialisasi <i>Network Monitoring</i>	81
Gambar 5.18. Tampilan Antarmuka <i>Network Monitoring</i>	82
Gambar 5.19. Grafik Hasil Uji Performa <i>Network Reachable</i> tanpa <i>Load Switching</i>	84
Gambar 5.20. Grafik Hasil Uji Performa <i>Network Reachable</i> dengan <i>Load Switching</i> tanpa waktu tunggu.....	85
Gambar 5.21. Grafik Hasil Uji Performa <i>Network Reachable</i> dengan <i>Load Switching</i> menggunakan waktu tunggu.....	87
Gambar 5.22. Grafik Hasil Uji Performa <i>Round Trip Time</i> tanpa menggunakan <i>Load Switching</i>	88
Gambar 5.23. Grafik Hasil Uji Performa <i>Round Trip Time</i> dengan <i>Load Switching</i> tanpa waktu tunggu.....	90
Gambar 5.24. Grafik Hasil Uji Performa <i>Round Trip Time</i> dengan <i>load switching</i> menggunakan waktu tunggu.....	92

Gambar 5.25. Grafik Hasil Uji Performa Pengunduhan tanpa <i>Load Switching</i>	94
Gambar 5.26. Grafik Hasil Uji Performa Pengunduhan dengan <i>Load Switching</i> tanpa waktu tunggu.....	95
Gambar 5.27. Grafik Hasil Uji Performa Pengunduhan dengan <i>Load Switching</i> menggunakan waktu tunggu.....	98
Gambar 5.29. Grafik Hasil Uji Performa Pengunggahan dengan <i>Load Switching</i> tanpa waktu tunggu.....	99
Gambar 5.30. Grafik Hasil Uji Performa Pengunggahan dengan <i>Load Switching</i> menggunakan waktu tunggu.....	101

DAFTAR TABEL

Tabel 1. Perangkat Keras Uji Coba Sistem.....	59
Tabel 2. Prosedur Uji Coba Inisialisasi <i>Controller</i>	64
Tabel 3. Uji Coba Inisialisasi <i>load switching</i>	66
Tabel 4. Uji Coba Pengambilan Statistik <i>Throughput</i>	68
Tabel 5. Uji Coba Memasukkan Data <i>Throughput</i> ke <i>Database</i>	70
Tabel 6. Uji Coba Proses Penentuan <i>Flow</i>	72
Tabel 7. Uji Coba Proses <i>Dial-Up</i>	74
Tabel 8. Uji Coba Proses <i>Forwarding</i> Paket.....	76
Tabel 9. Uji Coba Pengambilan dan Pengiriman <i>Throughput</i> ...	77
Tabel 10. Uji Coba Inisialisasi <i>Network Monitoring</i>	80
Tabel 11. Tabel <i>Delay Time</i> pada Uji Coba <i>Network Reachable</i> tanpa <i>Load Switching</i>	84
Tabel 12. Tabel <i>Delay Time</i> pada Uji Performa <i>Network</i> <i>Reachable</i> dengan <i>Load Switching</i> tanpa Waktu Tunggu.....	86
Tabel 13. Tabel <i>Delay Time</i> pada Uji Performa <i>Network</i> <i>Reachable</i> dengan <i>Load Switching</i> menggunakan Waktu Tunggu.....	87
Tabel 14. Tabel <i>Delay Time</i> pada Uji Coba <i>Round Trip Time</i> tanpa <i>Load Switching</i>	89
Tabel 15. Tabel <i>Delay Time</i> pada Uji Performa <i>Round Trip Time</i> dengan <i>Load Switching</i> tanpa Waktu Tunggu.....	91
Tabel 16. Tabel <i>Delay Time</i> pada Uji Performa <i>Round Trip Time</i> menggunakan <i>Load Switching</i> dengan Waktu Tunggu.....	92
Tabel 17. Tabel <i>Delay Time</i> pada Uji Performa Pengunduhan tanpa <i>Load Switching</i>	94
Tabel 18. Tabel <i>Delay Time</i> pada Uji Performa Pengunduhan dengan <i>Load Switching</i> tanpa Waktu Tunggu.....	96
Tabel 19. Tabel <i>Delay Time</i> pada Uji Performa Pengunduhan dengan <i>Load Switching</i> menggunakan Waktu Tunggu.....	97
Tabel 20. Tabel <i>Delay Time</i> pada Uji Performa Pengunggahan tanpa <i>Load Switching</i>	99

Tabel 21. Tabel <i>Delay Time</i> pada Uji Performa Pengunggahan dengan <i>Load Switching</i> tanpa Waktu Tunggu.....	100
Tabel 22. Tabel <i>Delay Time</i> pada Uji Performa Pengunggahan dengan <i>Load Switching</i> menggunakan <i>Waktu Tunggu</i>	101
Tabel 23. Tabel Rata-rata <i>Delay Time</i> pada Uji Performa tanpa <i>Load Switching</i>	102
Tabel 24. Tabel Rata-rata <i>Delay Time</i> pada Uji Performa menggunakan <i>Load Switching</i> tanpa Waktu Tunggu.....	102
Tabel 25. Tabel Rata-rata <i>Delay Time</i> pada Uji Performa menggunakan <i>Load Switching</i> tanpa Waktu Tunggu.....	103

DAFTAR KODE SUMBER

Kode Sumber 1. Kode Sumber Pengambilan Data <i>Host</i> pada OpenFlow <i>Router</i>	119
Kode Sumber 2. Kode Sumber Inisialisasi <i>Master Node</i>	120
Kode Sumber 3. Kode Sumber Pengambilan Data <i>Throughput</i> pada Tabel <i>Port Controller</i>	120
Kode Sumber 4. Kode Sumber Memasukkan Data <i>Throughput</i> kedalam <i>Database</i>	121
Kode Sumber 5. Kode Sumber Pengambilan Data <i>Throughput</i> Terbaik pada <i>Database</i> tanpa Waktu Tunggu.....	121
Kode Sumber 6. Kode Sumber Perbandingan dan Pembuatan <i>Flow</i>	121
Kode Sumber 7. Kode Sumber Pengambilan Data <i>Throughput</i> Terbaik pada <i>Database</i> dengan Waktu Tunggu.....	123
Kode Sumber 8. Kode Sumber Pengambilan <i>Round Trip Time</i>	123
Kode Sumber 9. Kode Program Pengambilan <i>Throughput Modem</i>	124
Kode Sumber 10. Kode program penghitungan dan pengiriman parameter.....	124

[Halaman ini sengaja dikosongkan]

BAB I PENDAHULUAN

Bab ini menjelaskan tentang hal-hal yang mendasari pengerjaan Tugas Akhir ini, antara lain latar belakang, rumusan masalah, tujuan pembuatan Tugas Akhir, manfaat pembuatan Tugas Akhir, metodologi, dan sistematika penulisan buku Tugas Akhir.

1.1. Latar Belakang

Perkembangan teknologi informasi yang sangat pesat dapat merubah pola hidup manusia dalam menjalani aktivitas sehari-hari. Salah satu perkembangan teknologi informasi yang merubah pola hidup manusia adalah perkembangan *internet*. *Internet* dibutuhkan karena hampir semua aktivitas manusia pada era modern ini menuntut mobilisasi yang tinggi. Selain dituntut untuk mobilisasi yang tinggi, manusia pada era modern ini juga dituntut untuk *multitasking*, artinya dapat melakukan beberapa tugas atau aktivitas dalam waktu yang bersamaan. Oleh karena itu, dibutuhkan koneksi *internet* untuk memenuhi kebutuhan mobilisasi dan *multitasking* yang dapat dinikmati kapanpun dan dimanapun meski sedang dalam perjalanan.

Kebutuhan akses *internet* yang tinggi menyebabkan *Internet Service Provider* (ISP) berlomba-lomba untuk memberikan pelayanan yang terbaik bagi pelanggan mereka dengan berbagai infrastruktur yang dimiliki. Salah satu infrastruktur yang diberikan kepada pelanggan agar dapat menikmati *internet* adalah melalui jaringan seluler (*Cellular Network*). Jaringan seluler menggunakan *Base Transceiver Station* (BTS) sebagai titik akses. Teknologi pada jaringan seluler yang berkembang saat ini adalah teknologi 4G (*Fourth Generation*). Teknologi 4G menggunakan *Long-Range Base Technologies* yang memungkinkan pengguna untuk melakukan akses kepada layanan yang berbeda, peningkatan jangkauan akses, dan memberikan akses yang lebih terpercaya (aspek keamanan). Layanan

4G akan memberikan akses data melalui beberapa teknologi berbasis IP dan menawarkan beberapa *bit rates* sampai dengan 50 Mbps [1]. Meskipun teknologi jaringan seluler yang sudah dikembangkan saat ini sudah mendukung kecepatan akses yang tinggi dan jangkauan yang lebih luas, perangkat komunikasi yang menggunakan jaringan seluler harus berada pada range BTS agar dapat berkomunikasi satu sama lain. Apabila sebuah perangkat yang menggunakan jaringan seluler berada pada sebuah wilayah yang tidak terjangkau oleh sinyal dari BTS dengan baik maka perangkat tersebut tidak dapat melakukan komunikasi secara maksimal. Wilayah yang memiliki sinyal rendah tersebut sering dijumpai ketika berada di daerah pelosok dimana jumlah tower BTS masih sangat sedikit.

Akses *internet* dibutuhkan untuk menyelesaikan berbagai aktivitas manusia. Untuk menunjang aspek mobilisasi dan *multitasking* maka tidak jarang seseorang membutuhkan akses *internet* kapan saja dan dimana saja, meskipun sedang dalam perjalanan. Ketika berada di dalam suatu perjalanan khususnya perjalanan darat (kereta api, bus, maupun kendaraan pribadi) seringkali terdapat suatu wilayah yang tidak dapat dijangkau oleh jaringan seluler. Hal ini menyebabkan pengguna tidak dapat mengakses suatu informasi melalui jaringan dan tidak memenuhi persyaratan *Quality of Service* (QOS). Agar memenuhi persyaratan *Quality of Service*, salah satu poin yang harus dipenuhi adalah suatu arsitektur jaringan tersebut dapat menangani *bandwidth* yang berubah-ubah (*variable bandwidth*) setiap saat [1]. Salah satu metode yang dapat digunakan untuk menangani hal ini adalah dengan melakukan *load switching* untuk pemilihan bandwidth tertinggi pada beberapa ISP menggunakan *Software Defined Network* (SDN) yang di implementasikan pada teknologi WLAN. *Load switching* yang dimaksud adalah pemilihan jalur aliran data dipilih berdasarkan kondisi *resource* yang terbaik dengan menggunakan parameter tertentu.

Software Defined Network (SDN) dipilih sebagai implementasi sistem karena SDN menyediakan fasilitas *Application Programming Interface* (API). API yang tersedia pada SDN dapat mengetahui statistik suatu perangkat dan memberikan kebebasan *programmer* untuk mengembangkan sebuah aplikasi pada perangkat tersebut. Infrastruktur SDN juga memungkinkan sebuah arsitektur jaringan tidak terpaku pada perangkat dengan *vendor* tertentu, karena SDN menggunakan standar protokol OpenFlow untuk berkomunikasi dengan sebuah perangkat.

1.2. Rumusan Permasalahan

Rumusan masalah yang akan dibahas dalam tugas akhir ini dapat disebutkan sebagai berikut :

1. Bagaimana program dapat melakukan deteksi *dynamic network* secara *realtime*?
2. Bagaimana program dapat melakukan pemilihan koneksi terbaik ketika terjadi perubahan *bandwidth*?
3. Bagaimana program dapat diimplementasikan dengan *software defined network*?
4. Bagaimana program dapat melakukan manajemen *client* yang terkoneksi ke sistem dan bersifat sementara?
5. Bagaimana program dapat melakukan manajemen koneksi yang sedang berjalan?

1.3. Batasan Masalah

Permasalahan yang dibahas dalam tugas akhir ini memiliki beberapa batasan antara lain:

1. Program yang dibuat menggunakan bahasa pemrograman java dengan memanfaatkan REST API.
2. Framework yang digunakan adalah OpenDaylight Helium-SR4.
3. Algoritma yang digunakan untuk pemilihan ISP adalah algoritma *least connections*.

4. ISP yang dilakukan *load switching* menggunakan 4 ISP GSM (Telkomsel, Indosat Ooredoo, XL, 3).
5. Arsitektur jaringan yang digunakan menggunakan 1 *controller*, 4 *modem* GSM, 4 *raspberrypi*, dan 1 *router*.
6. Parameter yang menjadi acuan untuk pemilihan jalur koneksi adalah *roud trip time* (RTT), *received throughput*, *send throughput*, dan *network reachable*.
7. Konfigurasi IP *address* pada pengguna menggunakan IP *address* statis.

1.4. Tujuan

Tujuan dari pembuatan tugas akhir ini antara lain:

1. Membuat sebuah aplikasi yang dapat meningkatkan *network reliability* pada jaringan dinamis.
2. Melakukan sebuah manajemen *client* yang terkoneksi dengan sistem dan bersifat sementara.
3. Melakukan manajemen koneksi dan *report*.

1.5. Manfaat

Manfaat dari pembuatan tugas akhir ini antara lain:

1. Memberikan layanan koneksi *internet* yang stabil secara realtime ketika berpindah dari suatu lokasi ke lokasi yang lain.
2. Menjadikan informasi dapat disampaikan kapanpun dan dimanapun dengan gangguan jaringan yang sekecil mungkin.

1.6. Metodologi

Adapun langkah-langkah yang ditempuh dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir

Tahap awal untuk memulai pengerjaan Tugas Akhir ini adalah penyusunan proposal Tugas Akhir. Proposal tugas akhir berisi tentang deskripsi pendahuluan dari tugas akhir mengenai

implementasi *Load Switching* pada jaringan seluler menggunakan *Software Defined Network* yang akan dibuat. Pendahuluan ini terdiri atas hal yang menjadi latar belakang diajukannya usulan tugas akhir, rumusan masalah yang diangkat, batasan masalah untuk tugas akhir, tujuan dari pembuatan tugas akhir, dan manfaat dari hasil pembuatan tugas akhir. Selain itu dijabarkan pula tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan tugas akhir. Sub bab metodologi berisi penjelasan mengenai tahapan penyusunan tugas akhir mulai dari penyusunan proposal hingga penyusunan buku tugas akhir. Terdapat pula sub bab jadwal kegiatan yang menjelaskan jadwal pengerjaan tugas akhir.

2. Studi Literatur

Tahap ini merupakan tahap pencarian informasi untuk pembelajaran teori-teori pendukung Tugas Akhir ini.

Pada studi literatur ini, akan dipelajari sejumlah referensi yang diperlukan dalam pembuatan aplikasi yang berasal dari paper, tutorial, dan publikasi ilmiah lainnya. Materi yang dijadikan referensi yaitu mengenai *Wireless Quality of Service*, *load switching/load balancing*, jaringan GSM, *software defined network*, *dynamic network area*, dan *Wireless Local Area Network (WLAN)*.

3. Analisis dan Perancangan Sistem

Tahap ini merupakan analisis dan perancangan sistem berdasarkan studi literatur dan mempelajari konsep sistem dan aplikasi yang akan dibuat. Dengan berbekal teori, metode, dan informasi yang sudah terkumpul pada tahap sebelumnya, diharapkan dapat membantu proses perancangan sistem.

4. Implementasi Sistem

Tahap ini merupakan implementasi rancangan sistem yang telah dibuat. Tahap ini merealisasikan apa yang terdapat pada tahapan perancangan yang telah dibuat sebelumnya, sehingga menjadi sebuah rancang bangun sistem yang sesuai dengan apa yang

telah direncanakan.

Implementasi sistem yang akan dibuat akan menggunakan sistem jaringan riil yaitu menggunakan 4 buah modem GSM dengan *interface* USB, 4 buah Raspberrypi, dan sebuah *server* dengan menggunakan *controller* OpenDaylight.

5. Pengujian dan Evaluasi

Aplikasi akan diuji setelah selesai diimplementasikan menggunakan skenario yang sudah dipersiapkan. Pengujian pada Tugas Akhir ini akan dipisahkan menjadi 2 tahap, yaitu:

1. Pengujian fungsionalitas, pengujian ini meliputi pengujian fungsi-fungsi pokok sistem agar dapat menjalankan fungsi-fungsi utama pada sistem. Pengujian fungsionalitas meliputi inisialisasi *controller*, onisialisasi *load switching*, pengambilan statistik *throughput*, memasukkan data ke *database*, penentuan *flow*, proses *dial-up*, *forwarding* paket, pengambilan dan pengiriman *throughput* dan inisialisasi *network monitoring*.
2. Pengujian performa, pengujian performa digunakan untuk mengetahui apakah sistem dapat menjalankan fungsi utama dengan baik. Pada pengujian ini akan dihitung nilai-nilai performa yang dibutuhkan, meliputi nilai performa sebelum menggunakan sistem *load switching* dan setelah menggunakan sistem *load switching*. Parameter yang dijadikan nilai acuan adalah *network reachable*, *round trip time*, *throughput*, dan *latency* yang diperoleh oleh *client*.

1.7. Sistematika Penulisan Tugas Akhir

Buku Tugas Akhir ini disusun dengan sistematika penulisan seperti yang tertera di bawah sehingga diharapkan dapat berguna bagi pembaca yang tertarik untuk melanjutkan pengembangan. Antara lain sebagai berikut:

BAB I. PENDAHULUAN

Bab ini berisi latar belakang, rumusan permasalahan, tujuan, manfaat, batasan permasalahan, metodologi, dan sistematika penulisan.

BAB II. TINJAUAN PUSTAKA

Bab ini berisi dasar teori yang mendukung dalam pembuatan Tugas Akhir ini.

BAB III. ANALISIS DAN PERANCANGAN SISTEM

Bab ini berisi tentang perancangan sistem yang terdiri dari perancangan perangkat lunak yang digambarkan dalam diagram alir, perancangan arsitektur jaringan dalam diagram arsitektur.

BAB IV. IMPLEMENTASI

Bab ini membahas implementasi dari perancangan perangkat lunak dan perancangan arsitektur jaringan yang dibahas pada bab sebelumnya.

BAB V. UJI COBA DAN EVALUASI

Bab ini menjelaskan pengujian perangkat lunak dengan melakukan serangkaian pengujian seperti pengujian fungsionalitas, dan uji performa. Pengujian akan dilakukan dengan menggunakan jaringan riil dan menggunakan transportasi darat agar mendapatkan kondisi jaringan secara dinamis, kemudian mengevaluasi konfigurasi yang dipakai untuk selanjutnya agar didapatkan hasil terbaik.

BAB VI. PENUTUP

Bab ini menyampaikan kesimpulan dari hasil uji coba yang dilakukan dan membahas saran untuk pengembangan lebih lanjut.

[Halaman ini sengaja dikosongkan]

BAB II TINJAUAN PUSTAKA

Bab ini menjelaskan teori-teori yang berhubungan dengan sistem yang diimplementasikan. Hal ini bertujuan untuk memberikan gambaran umum terhadap sistem yang dibuat dan berguna untuk menunjang pembuatan sistem sehingga kebutuhannya dapat diketahui.

2.1. *Wireless Quality of Service*

Quality of Service (QOS) dalam konteks jaringan nirkabel (*wireless*) adalah salah satu dimensi mobilitas, yaitu suatu hal yang dapat membedakan aplikasi *mobile* berbeda dengan kondisinya ketika diam [1]. QOS diperlukan sebagai sebuah metode untuk memenuhi kriteria pelayanan sistem bagi pengguna, yaitu *confidentiality*, *integrity*, dan *availability*. Beberapa aspek yang menjadi topik utama dalam QOS adalah *failure and recovery mechanism*, *variable bandwidth*, *computing distribution*, *discovery mechanism*, *variable latency*, dan *performance feedback*. Solusi yang diberikan berdasarkan problem dari beberapa aspek tersebut berbeda-beda tergantung dari arsitektur yang kita tentukan untuk mengimplementasikan sistem yang kita buat.

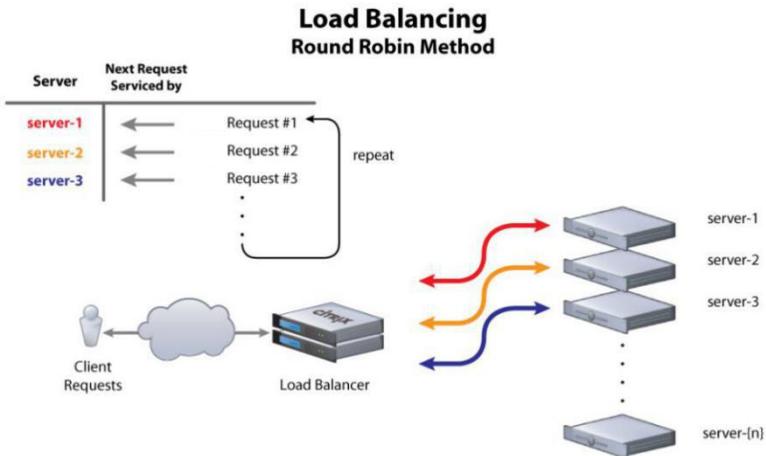
2.2. *Load Switching*

Load Switching adalah sebuah metode yang termasuk dalam metode *load balancing* yang berfungsi untuk mendistribusikan lalu lintas data diantara beberapa sumber daya *server* [2]. Sebuah perangkat atau aplikasi yang memiliki fungsi untuk menjalankan metode *load balancing* disebut dengan *load balancer*. *Load balancer* bertindak sebagai pengatur lalu lintas yang menerima permintaan lalu lintas yang datang. Kemudian *load balancer* meneruskannya kepada semua *server* yang mampu menangani permintaan tersebut dengan cara memaksimalkan kecepatan dan kapasitas dari pendayagunaan *server* dan memastikan bahwa tidak ada satupun *server* yang bekerja terlalu banyak (*overworked*) sehingga

menyebabkan performa *server* menurun. Implementasi *load balancer* memiliki fungsi yang beragam, diantaranya adalah sebagai *load switching*, *traffic engineering*, dan *intelligent traffic switching*.

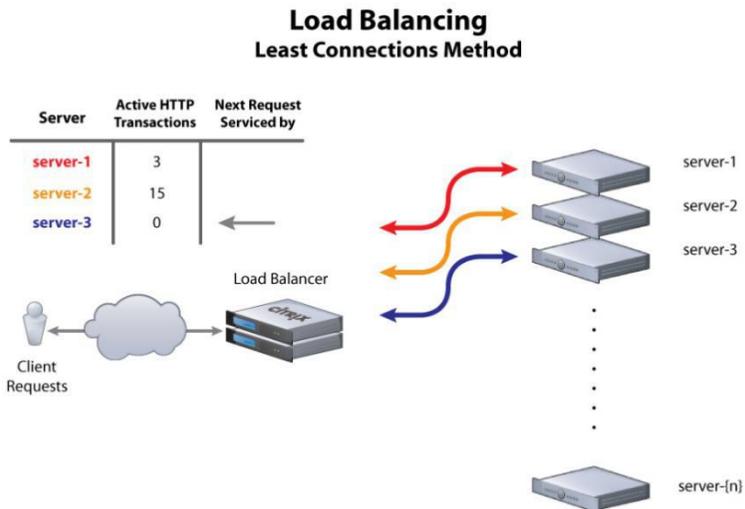
Algoritma yang digunakan pada metode *load balancing* sangat bervariasi, tergantung dengan kebutuhan sistem dan permasalahan yang akan ditangani. Algoritma yang diterapkan pada suatu sistem atau perangkat bisa jadi berbeda dengan sistem atau perangkat yang lain. Beberapa algoritma *load balancing* yang sering digunakan yaitu:

1. *Round Robin*, yaitu sebuah metode pembagian lalu lintas data yang masuk dengan cara mendistribusikan kepada seluruh kelompok *server* secara berurutan. Sehingga terbentuk suatu putaran atau urutan *server* yang dapat melayani permintaan secara merata [3]. Secara umum, algoritma *round robin* ditunjukkan pada gambar 2.1 berikut ini.



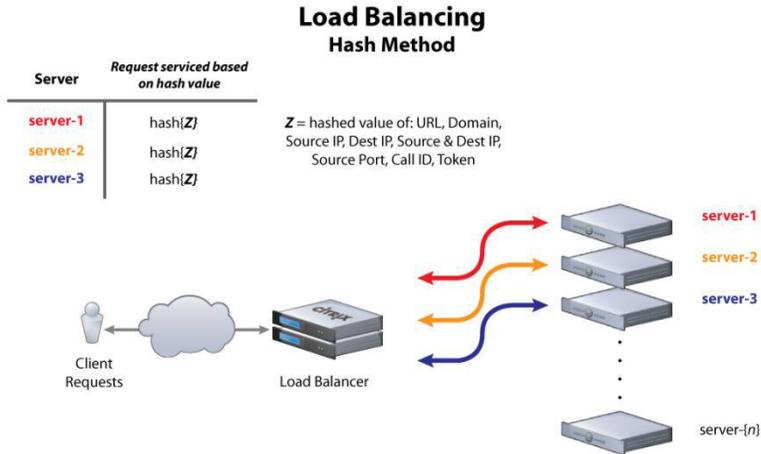
Gambar 2.1. Algoritma Round Robin [3]

2. *Least Connection*, yaitu sebuah metode pendistribusian lalu lintas data yang masuk dengan cara mengirimkan permintaan yang baru kepada *server* dengan koneksi aktif ke klien yang paling sedikit [4]. Kapasitas komputasi dari setiap server dapat berbeda-beda dalam menentukan banyaknya koneksi kepada klien. Secara umum, algoritma *least connection* ditunjukkan pada gambar 2.2 berikut ini.



Gambar 2.2. Algoritma *Least Connection* [4]

3. *Hash*, yaitu sebuah metode pendistribusian lalu lintas data dengan cara menghitung nilai *hash* kemudian mengirimkan permintaan ke *server*. Algoritma *hash* hampir mirip dengan algoritma *persistent load balancing*, dimana suatu koneksi dari *client* yang telah memiliki session diteruskan kepada *server* yang selalu sama [5]. Secara umum, algoritma hash dapat ditunjukkan pada gambar 2.3 berikut ini.

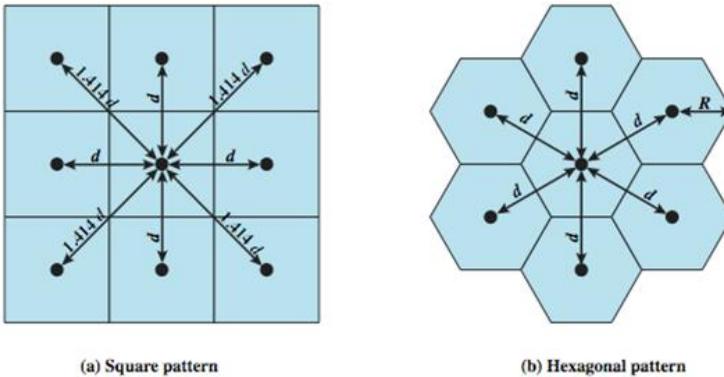


Gambar 2.3. Algoritma Hash [5]

2.3. Jaringan Seluler

Jaringan seluler (*Cellular Network*) adalah sebuah jaringan telekomunikasi yang menggunakan jaringan radio dan menggunakan beberapa pemancar yang berdaya rendah (*multiple low-power transmitter*) kurang dari 100 W [6]. Karena sistem pemancar yang digunakan memiliki daya yang rendah, maka jaringan seluler dibagi menjadi beberapa area. Setiap area pada jaringan seluler dibagi menjadi beberapa sel (*cell*), setiap sel akan dilayani oleh sebuah antena dan akan dialokasikan pada frekuensi tertentu yang dilayani oleh *base station*. Sebuah *base station* terdiri dari *transmitter*, *receiver*, dan *control unit*.

Setiap *base station* yang berdekatan dapat berkomunikasi dan akan membentuk sebuah *pattern* yang menyelimuti area tersebut. Pattern yang digunakan biasanya membentuk *square pattern* atau *hexagonal pattern* seperti yang ditunjukkan pada gambar 2.4 geometri seluler berikut ini.



Gambar 2.4. Geometri Jaringan Seluler [6]

Teknologi pada jaringan seluler yang berkembang saat ini adalah teknologi 4G (*Fourth Generation*). Teknologi 4G menggunakan *Long-Range Base Technologies* yang memungkinkan pengguna untuk melakukan akses kepada layanan yang berbeda, peningkatan jangkauan akses, dan memberikan akses yang lebih terpercaya (aspek keamanan). Layanan 4G akan memberikan akses data melalui beberapa teknologi berbasis IP dan menawarkan beberapa bit rates sampai dengan 50 Mbps [1].

2.4. *Software Defined Network*

Software Defined Network (SDN) merupakan sebuah arsitektur jaringan yang dinamis, dapat dikelola (*manageable*), hemat biaya, dan mudah beradaptasi dengan perubahan kondisi jaringan [7].

Secara umum, arsitektur *software defined network* adalah sebagai berikut:

- ***Directly programmable***, kontrol pada jaringan dapat diprogram secara langsung karena dipisahkan dari fungsi *forwarding*.

- **Agile**, lapisan kontrol abstrak dari fungsi *forwarding* dapat memungkinkan seorang administrator untuk mengatur sebuah lalu lintas data (*flow*) pada jaringan yang besar secara dinamis dan sesuai dengan kebutuhan.
- **Centrally managed**, sistem pengatur sebuah jaringan dipusatkan pada *software-based SDN Controllers* dan hanya dapat terlihat oleh aplikasi dan *policy* sebagai sebuah *logical switch*.
- **Programmatically configured**, SDN memberikan administrator jaringan untuk melakukan konfigurasi, manajemen, pengamanan, dan optimisasi sebuah jaringan secara cepat dan dinamis melalui program SDN yang otomatis.
- **Open standards-based and vendor-neutral**, SDN memberikan keleluasaan penggunaan karena SDN menyederhanakan dan melakukan standarisasi sebuah desain jaringan dan operasi jaringan yang secara khusus telah ditentukan oleh suatu merk tertentu.

2.4.1. Controller

Saat buku Tugas Akhir ini ditulis, terdapat banyak ragam *controller* yang dapat mendukung arsitektur *software defined network*, diantaranya adalah POX, RYU, Cisco SDN, HP SDN, OpenDaylight, dan lain-lain.

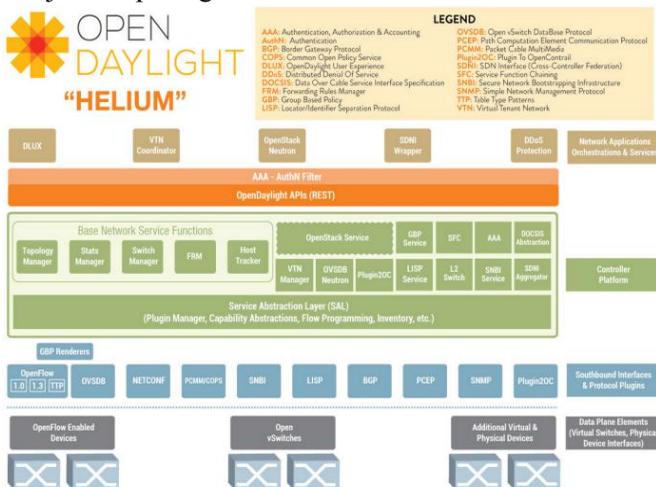
OpenDaylight *Controller* adalah sebuah *Java Virtual Machine* (JVM) software yang dapat berjalan di semua sistem operasi dan hardware selama mendukung Java [8]. Controller yang digunakan merupakan implementasi dari konsep *Software Defined Network* (SDN) dan menggunakan beberapa tools yaitu :

1. Maven, maven digunakan untuk mempermudah membangun otomatisasi.
2. OSGi, *framework* ini merupakan *back-end* dari OpenDaylight yang memungkinkan untuk memuat bundel secara dan paket

JAR *file* secara dinamis, kemudian mengikat bundel bersama untuk bertukar informasi.

3. Java *interfaces*, Java *interfaces* digunakan untuk *event listening*, *spesifikasi*, dan pembentukan *pattern*.
4. REST API, *Representational State Transfer Application Program Interface* (REST API) digunakan untuk menghubungkan OpenDaylight dengan aplikasi yang dibangun, misalnya sebagai manajemen topologi, *host tracker*, *flow programmer*, *routing*, dll.

OpenDaylight mendukung beberapa fitur yang penting dalam pengembangan *Software Defined Network*, antara lain : *Microservices Architecture*, *Multiprotocol Support*, *Security*, *Scalability*, *Stability*, dan *Performance*, serta memiliki arsitektur yang ditunjukkan pada gambar 2.5 berikut ini.



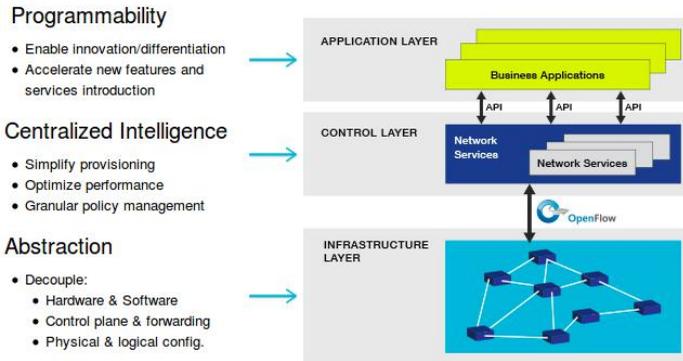
Gambar 2.5. Arsitektur *Controller* OpenDaylight Helium [9]

2.4.2. OpenFlow

OpenFlow adalah sebuah standar komunikasi pada jaringan yang mendefinisikan antara lapisan kontrol dan lapisan *forwarding*

dari sebuah arsitektur SDN. OpenFlow memungkinkan akses langsung dan melakukan manipulasi dari *forwarding plane* pada sebuah perangkat jaringan seperti *switch* dan *router*, baik secara fisik maupun virtual (*hypervisor-based*).

Beberapa fitur yang ditawarkan oleh OpenFlow digambarkan oleh gambar 2.6 sebagai berikut:



Gambar 2.6. Fitur OpenFlow [8]

2.4.3. REST (*Representational State Transfer*) API

REST API adalah sebuah abstraksi dari elemen pada arsitektur yang terdapat pada sistem *hypermedia* terdistribusi [10]. REST API pada aplikasi *web service* menerapkan konsep perpindahan antar *state*. *State* dapat digambarkan seperti *browser* ketika meminta suatu halaman *web*, maka *server* akan mengirimkan *state* halaman *web* yang sekarang ke *browser*. Bernavigasi melalui *link-link* yang disediakan sama halnya dengan mengganti *state* dari halaman *web*. Begitu pula REST bekerja, dengan bernavigasi melalui *link-link* HTTP untuk melakukan aktivitas tertentu, seakan-akan terjadi perpindahan *state* satu sama lain. Perintah HTTP yang bisa digunakan adalah fungsi GET, POST, PUT, dan DELETE. Proses pengiriman dan penerimaan data dalam bentuk XML tanpa ada protokol pemaketan data, sehingga informasi yang diterima lebih mudah dibaca dan dilakukan *parsing* di sisi *client*.

Dalam pengaplikasiannya, REST lebih banyak digunakan untuk *web service* yang berorientasi pada *resource*. Maksudnya orientasi pada *resource* adalah orientasi yang menyediakan *resource-resource* sebagai layanannya dan bukan kumpulan dari aktivitas yang mengolah *resource* itu. Beberapa contoh *web service* yang menggunakan REST adalah : Flickr API, YouTube API, dan Amazon API.

2.5. *Dynamic Network Area*

Dynamic network adalah suatu kondisi dimana jaringan yang diterima pada sebuah perangkat mengalami perubahan kenaikan atau penurunan secara signifikan ketika berada di suatu wilayah tertentu. *Dynamic network* biasanya ditemui ketika sebuah perangkat berpindah dari satu tempat ke tempat lainnya. Ketika sebuah perangkat berada pada kondisi jaringan yang tidak stabil maka mengakibatkan data yang diterima atau dikirim tidak optimal, dan penggunaan baterai lebih banyak.

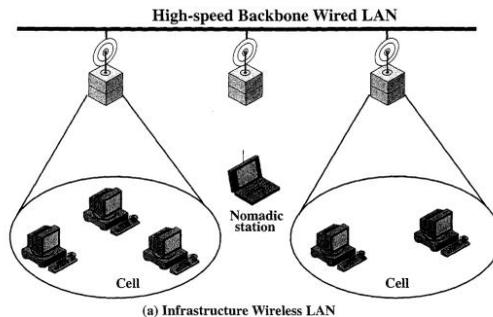
Salah satu *area* yang termasuk *area* paling buruk pada jaringan dinamis adalah *blank spot area*. *Blank spot area* atau dalam referensi lain disebut sebagai *dead zone*, *call drop*, dan lain-lain adalah sebuah *area* dimana jaringan seluler tidak dapat digunakan karena mengalami interferensi atau kondisi sinyal yang rendah (karena terlalu jauh dengan *tower* BTS) [6]. Akibat dari *blank spot area* ini adalah paket yang dikirimkan oleh *client* akan di *drop* oleh sistem, sehingga komunikasi dan pertukaran data tidak dapat dilakukan.

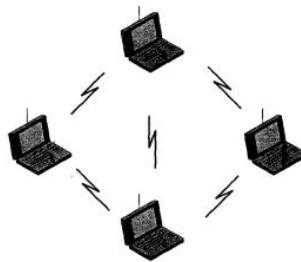
Pada jaringan *wireless* khususnya jaringan seluler terdapat 2 (dua) aspek inti yang sangat mempengaruhi komunikasi, yaitu kekuatan sinyal (*signal strength*) dan perambatan sinyal (*signal propagation*). Meskipun pada suatu daerah seorang *client* memperoleh sinyal yang tinggi, namun apabila perambatan sinyal lambat maka kecepatan yang akan didapat juga tidak maksimal. Pengaruh aktivitas manusia juga dapat menyebabkan gangguan dalam komunikasi

wireless. Misalnya, suara knalpot kendaraan bermotor yang memiliki frekuensi lebih tinggi dari frekuensi jaringan seluler di kota besar.

2.6. *Wireless Local Area Network*

Wireless Local Area Network (WLAN) adalah sebuah metode distribusi jaringan nirkabel untuk menghubungkan dua atau lebih perangkat jaringan dengan menggunakan gelombang radio berfrekuensi tinggi [11]. Sederhananya, WLAN mengombinasikan antara *Wired Local Area Network* dan *Mobile Data Network*. Pada gambar 2.7 berikut ini menunjukkan perbedaan antara LAN nirkabel yang mendukung ekstensi LAN dan kebutuhan akses nomaden dan nirkabel *ad hoc* LAN. Dalam kasus yang pertama, LAN nirkabel membentuk infrastruktur stasioner yang terdiri dari satu atau lebih sel dengan modul kontrol untuk setiap sel. Dalam sel, mungkin ada sejumlah sistem akhir stasioner. Stasiun nomaden dapat berpindah dari satu sel ke sel lain. Sebaliknya, tidak ada infrastruktur untuk jaringan *ad hoc*. Sebaliknya, koleksi rekan stasiun dalam jangkauan satu sama lain mungkin dinamis mengkonfigurasi diri ke jaringan sementara.

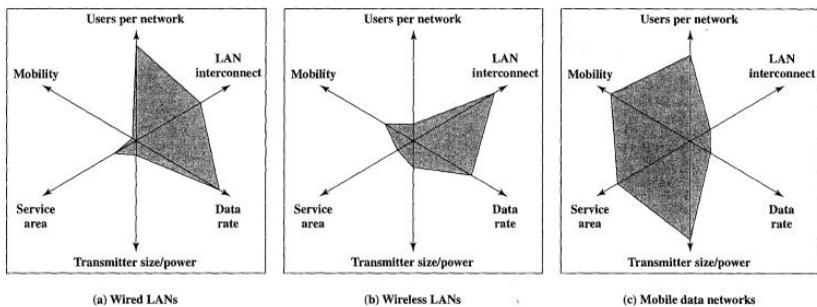




(b) Ad hoc LAN

Gambar 2.7. Konfigurasi *Wireless Local Area Network* [6]

Pada gambar 2.8 berikut ini menunjukkan perbandingan *Wired LAN*, *Wireless LAN*, dan *Mobile Data Network* menggunakan grafik Kivat.



Gambar 2.8. Perbandingan *Mobile Data Network* menggunakan Grafik Kivat [6]

[Halaman ini sengaja dikosongkan]

BAB III

ANALISIS DAN PERANCANGAN SISTEM

Bab ini secara khusus menjelaskan perancangan sistem yang dibuat dalam Tugas Akhir ini. Bab ini berisi deskripsi umum sistem, arsitektur sistem, perancangan arsitektur jaringan, perancangan proses beserta alurnya, dan perancangan antarmuka sistem *traffic monitoring*.

3.1. Deskripsi Umum Sistem

Sistem yang dibangun pada Tugas Akhir ini berupa sebuah aplikasi *load switching* yang diimplementasikan pada *software defined network (SDN)* dengan *controller* OpenDaylight. Aplikasi *load switching* yang dibuat diharapkan dapat beradaptasi dengan kondisi jaringan yang tidak stabil, khususnya jaringan seluler. Sehingga aplikasi dapat menyesuaikan dan meminimalisasi adanya gangguan ketika kondisi jaringan sedang tidak stabil. Aplikasi yang dibuat dengan memanfaatkan protokol OpenFlow agar aplikasi dapat berjalan dengan baik pada *software defined network*.

Secara umum, aplikasi akan menggunakan 4 (empat) *modem* sebagai jalur koneksi. Berdasarkan 4 (empat) perangkat modem tersebut akan diambil koneksi terbaik berdasarkan kualitas *throughput* yang dihasilkan oleh masing-masing koneksi per detik. Masing-masing data *throughput* tersebut kemudian dikirimkan ke *controller* untuk memodifikasi tabel *throughput port* (tx dan rx) pada tabel *port controller*. Setelah data *throughput* pada tabel *throughput port* pada *controller* dimodifikasi, maka aplikasi yang berada pada mesin *controller* akan memasukkan data tersebut ke *database* influxdb agar data tersimpan dan dapat dianalisis lebih lanjut.

Setelah data *throughput* dimasukkan ke database influxdb maka aplikasi akan menentukan jalur terbaik saat itu dengan menggunakan rumus sebagai berikut:

$$Rx_{bps} = Rx_2 - Rx_1 \quad (3.1)$$

$$Tx_{bps} = Tx_2 - Tx_1 \quad (3.2)$$

$$Parameter = \frac{Rx_{bps} + Tx_{bps}}{RTT} \quad (3.3)$$

Keterangan:

Rx_1 = Jumlah paket yang diterima pada detik pertama (*bytes*)

Rx_2 = Jumlah paket yang diterima pada detik kedua (*bytes*)

Tx_1 = Jumlah paket yang dikirim pada detik pertama (*bytes*)

Tx_2 = Jumlah paket yang dikirim pada detik kedua (*bytes*)

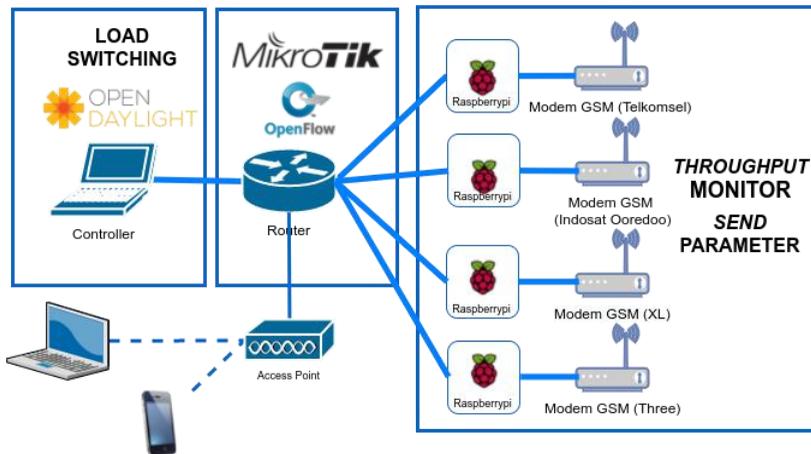
Rx_{bps} = Paket yang diterima per detik (*byte per second*)

Tx_{bps} = Paket yang dikirim per detik (*byte per second*)

RTT = *Round Trip Time* (waktu yang dibutuhkan sebuah paket untuk kembali ke pengirim) (*milliseconds*)

kemudian aplikasi akan melakukan pembuatan *flow* (jalur data) yang akan digunakan oleh *client* ke *internet*. Proses tersebut akan berlangsung terus menerus selama aplikasi dijalankan.

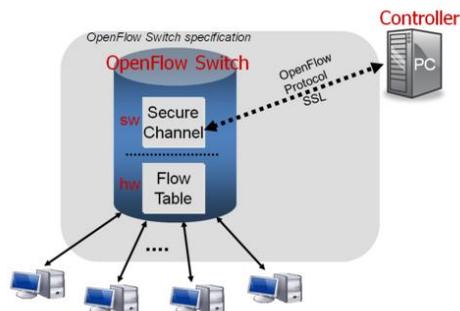
Arsitektur sistem secara umum ditunjukkan pada gambar 9 dibawah ini. Pada gambar 3.1 dibawah dapat dilihat bahwa mesin *controller* akan menjalankan 2 (dua) aplikasi yaitu *controller* OpenDaylight dan aplikasi *load switching*. *Router* akan menjalankan sistem operasi Mikrotik RouterOS dan menjalankan protokol OpenFlow. Sedangkan Raspberrypi akan menjalankan aplikasi *throughput* monitor dan melakukan pengiriman parameter *throughput*.



Gambar 3.1. Arsitektur Sistem secara Umum

3.2. Arsitektur Sistem

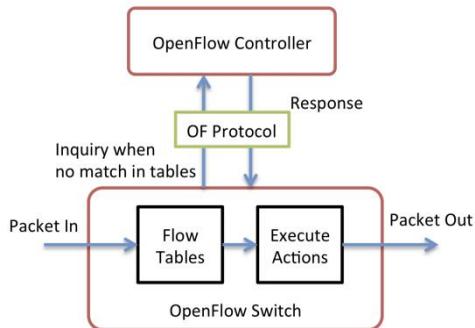
Rancangan arsitektur sistem pada Tugas Akhir ini dibuat dengan memanfaatkan protokol OpenFlow untuk berkomunikasi dengan *controller*. Protokol OpenFlow dirancang dengan memisahkan antara lapisan aplikasi (*application layer*), lapisan kontrol (*control layer*), dan lapisan infrastruktur (*infrastructure layer*) seperti yang digambarkan pada gambar 3.2. Rancangan pada implementasi Tugas Akhir ini disusun berdasarkan gambar berikut:



Gambar 3.2. Arsitektur Jaringan OpenFlow [12]

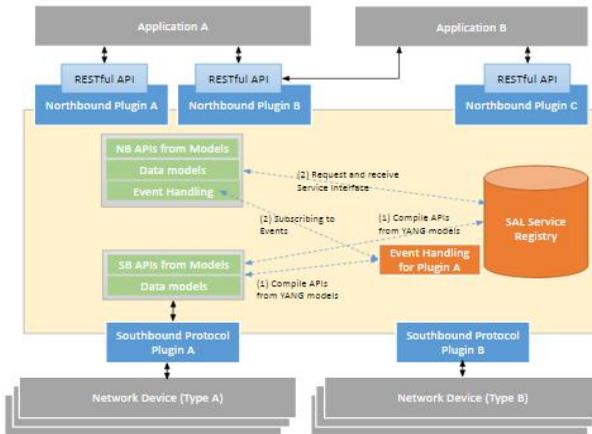
Dilihat dari gambar diatas dapat disimpulkan bahwa *controller* berkomunikasi dengan *switch* yang menerapkan protokol OpenFlow melalui jalur *secure channel (SSL)* untuk membuat sebuah *flow table* yang ditanamkan pada sebuah hardware.

Alur dari proses sebuah pertukaran data dimulai dari masuknya suatu paket ke dalam OpenFlow *switch*. OpenFlow *switch* akan mengirimkan data paket tersebut kepada *controller* untuk melakukan pengecekan pada *flow table* terhadap sebuah paket, apakah sudah terdapat data paket yang *match* dengan paket yang baru masuk. Kemudian *controller* akan memberikan respon terhadap paket yang masuk untuk membuat suatu perlakuan terhadap paket tersebut, apakah di-*drop* atau dibuatkan sebuah *flow table* baru berdasarkan paket yang masuk tersebut. Secara garis besar, alur dari proses penanganan sebuah paket digambarkan pada gambar 3.3 berikut ini:



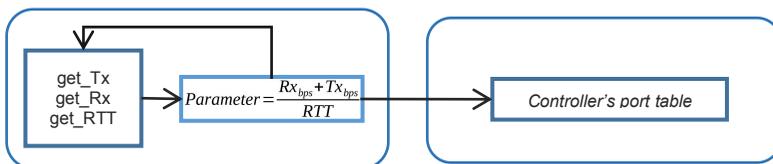
Gambar 3.3. Proses Penanganan Paket pada OpenFlow [13]

Aplikasi *load switching* akan berjalan dengan memanfaatkan REST API yang disediakan oleh *controller* dan berfungsi sebagai jembatan sebuah aplikasi untuk melakukan modifikasi *resource* yang disediakan oleh *controller*. Diagram cara kerja aplikasi dengan REST API ditunjukkan pada gambar 3.4 berikut ini:



Gambar 3.4. Diagram Cara Kerja REST API

Aplikasi yang berjalan pada *host router* secara periodik akan melakukan kalkulasi *throughput* dan mengirimkannya ke *host router* yang lain untuk melakukan modifikasi data *throughput port* pada lapisan kontrol. Sistem yang berjalan pada *host router* digambarkan pada gambar 3.5 berikut ini:



Gambar 3.5. Diagram Sistem Host Router

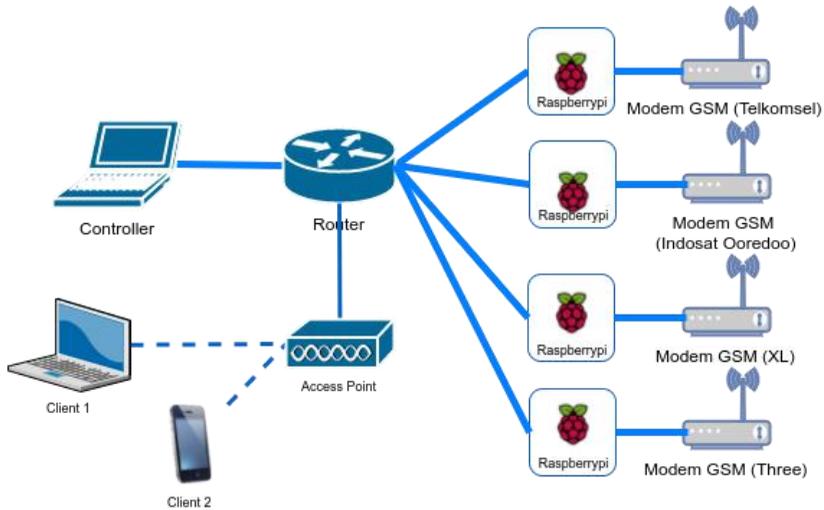
3.3. Perancangan Arsitektur Jaringan

Perancangan arsitektur jaringan pada Tugas Akhir ini menggunakan perangkat riil yang dimaksudkan untuk menguji coba apakah teori-teori dan algoritma pada *software defined network* dapat diimplementasikan dengan baik untuk aplikasi tertentu. Arsitektur jaringan yang dibuat dirancang agar sistem dapat menangani dan dapat beradaptasi dengan kondisi jaringan dinamis, sehingga dapat

menentukan apa yang harus dilakukan ketika kondisi tertentu (*decision making*). Secara teori, sudah banyak implementasi sistem yang serupa namun dengan menggunakan arsitektur jaringan *virtual* atau simulasi dengan menggunakan mininet sebagai infrastruktur dan linux sebagai sistem operasi pada mesin *controller*.

Sistem ini membutuhkan minimal 1 buah komputer yang difungsikan sebagai mesin *controller* dengan sistem operasi Linux maupun Windows dan secara langsung dapat terhubung satu *subnet* dengan sebuah perangkat yang memiliki dukungan protokol OpenFlow. Sistem ini menggunakan 4 buah modem dengan berbedabeda ISP sebagai *gateway* untuk melakukan koneksi terhadap *internet* dan 4 buah Raspberrypi sebagai *host router* yang berfungsi sebagai *dialer* masing-masing *modem*. Raspberrypi yang digunakan didesain agar terhubung secara langsung dengan perangkat yang dikontrol oleh *controller*. Karena pada Tugas Akhir ini menggunakan *router* Mikrotik sebagai perangkat yang dikontrol, maka konfigurasi *port* yang terhubung dengan *host router* harus dilakukan konfigurasi sebagai *master port*.

Perangkat *client* yang terhubung dengan sistem akan dihubungkan melalui *wireless access point* yang terdapat pada *interface router* Mikrotik yang sudah ditambahkan kedalam OpenFlow *port*. Konfigurasi alamat IP pada *client* menggunakan alamat IP statis, karena *controller* belum mendukung alamat IP secara dinamis. Rancangan arsitektur jaringan secara lebih jelas dapat dilihat pada gambar 3.6 berikut ini:



Gambar 3.6. Perancangan Arsitektur Jaringan

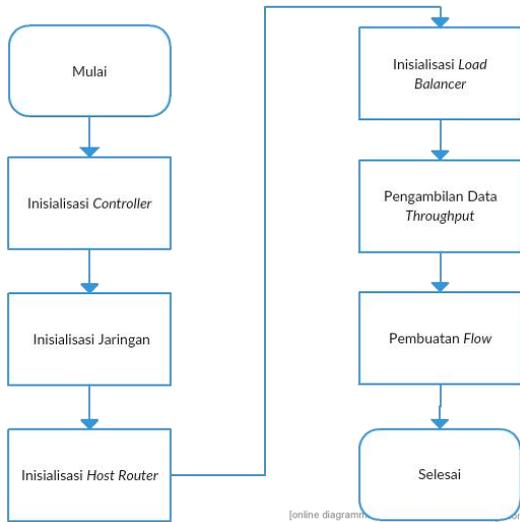
3.4. Diagram Alir Aplikasi

Alur kerja pada perancangan sistem *load switching* dengan menggunakan *Software Defined Network* ini dibagi menjadi 3 (tiga) bagian, yaitu alur kerja *controller*, alur kerja *host router*, dan alur kerja aplikasi *load switching*. Alur kerja *controller* adalah alur kerja yang dijalankan pada mesin *controller*, mulai dari *controller* dijalankan sampai selesai. Alur kerja *host router* adalah alur kerja beberapa modul yang berjalan pada *host router* yaitu pengambilan parameter *throughput* yang akan dikirimkan ke *controller*. Alur kerja aplikasi *load switching* yaitu alur kerja yang dijalankan pada mesin *controller* untuk menentukan jalur terbaik yang akan dilalui oleh sebuah paket.

3.4.1. Diagram Alir Inisialisasi Sistem

Saat pertama kali sistem dijalankan, sistem akan menjalankan *controller* OpenDaylight terlebih dahulu. *Controller* OpenDaylight digunakan sebagai *server* yang menunggu koneksi dari OpenFlow *switch*. Setelah *controller* selesai dijalankan, maka selanjutnya

adalah menjalankan *router* yang sudah dilakukan konfigurasi menjadi OpenFlow *switch*. Kemudian, *host router* beserta aplikasi pengambilan data *throughput* dapat dijalankan. Pada tahap ini, *host router* sudah dapat melakukan pengiriman data parameter kepada tabel *port* yang ada pada *controller*. Setelah aplikasi *load switching* dijalankan, maka aplikasi akan memanggil fungsi untuk pengambilan parameter kemudian melakukan pembuatan *flow*. Secara garis besar, diagram alir keseluruhan sistem yang sedang berjalan ditunjukkan pada gambar 3.7 berikut ini.

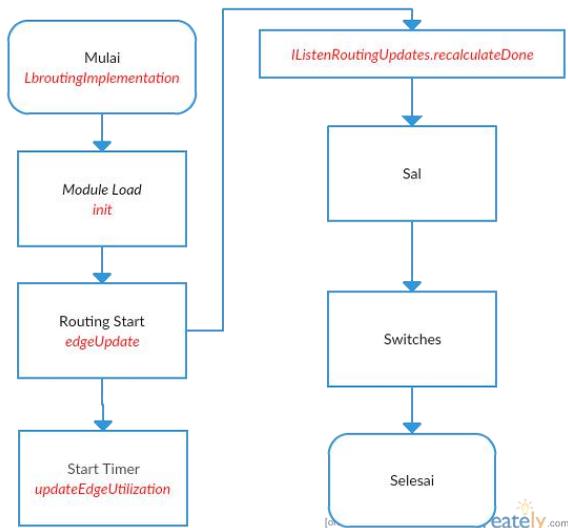


Gambar 3.7. Diagram Alir Inisialisasi Sistem

3.4.2. Diagram Alir *Controller*

Ketika *controller* dijalankan pertama kali, maka *controller* akan menjalankan sebuah *plugin LbroutingImplementation*, kemudian baru menjalankan inisialisasi modul. Setelah modul terinisialisasi, maka *controller* akan menjalankan protokol *routing* dan melakukan *update edge* yang saling terhubung secara periodik.

Kemudian *controller* akan menjalankan servis sal (*service abstraction layer*) dan kemudian melakukan *discovery* pada *switch*. Diagram alir *controller* ditunjukkan pada gambar 3.8 dibawah ini.

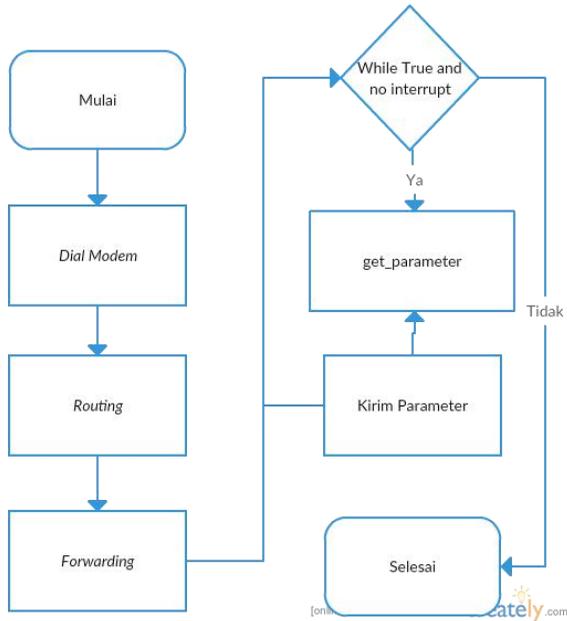


Gambar 3.8. Diagram Alir *Controller*

3.4.3. Diagram Alir *Host Router*

Proses yang terjadi pada *host router* berjalan secara terpisah dengan *controller*. Modul-modul yang dijalankan pada *host router* akan dijalankan secara otomatis pada saat pertamakali *host router* dinyalakan. Modul yang pertama kali dijalankan adalah *modem dialer*. Kemudian *host router* akan menjalankan *routing* dan *forwarding*. Setelah itu *host router* akan mengambil dan memroses data *throughput* beserta *round trip time* menjadi sebuah parameter yang kemudian dikirim kepada *controller* untuk memodifikasi tabel *port* pada *controller*. Proses pengambilan, pemrosesan, dan pengiriman parameter tersebut dijalankan terus menerus selama

sistem tidak mendapat *interrupt* maupun dimatikan. Diagram alir *host router* ditunjukkan pada gambar 3.9 berikut ini.



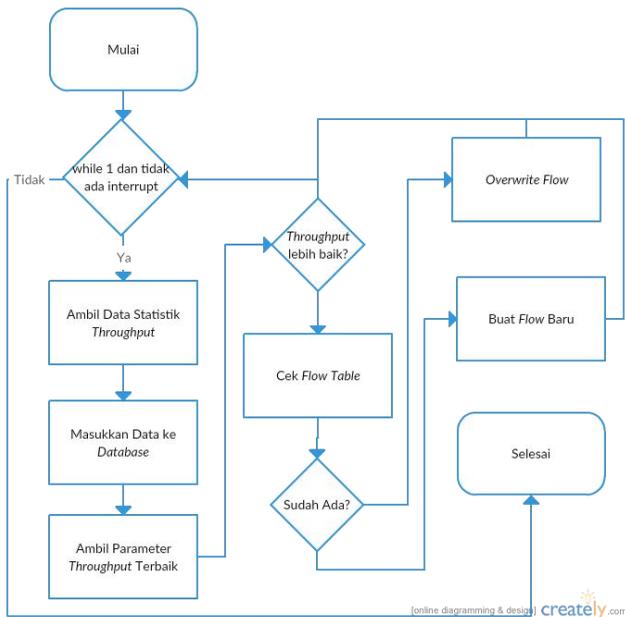
Gambar 3.9. Diagram Alir *Host Router*

3.4.4. Diagram Alir *Load Switching*

Aplikasi *load switching* dijalankan pada *controller* dengan modul terpisah. Oleh karena itu, aplikasi *load switching* ini dijalankan setelah *controller* sukses terinisialisasi. Aplikasi *load switching* akan melakukan pengambilan statistik *throughput* dari masing-masing *port* yang ada pada OpenFlow *switch*. Setelah data statistik *throughput* dapat diambil, maka aplikasi akan memasukkan data tersebut kedalam basis data influxdb. Kemudian aplikasi akan menjalankan *query* untuk mengambil data *throughput* terbaik saat ini. Apabila *throughput* saat ini lebih baik dari *throughput* sebelumnya,

maka aplikasi akan melakukan pengecekan terhadap *flow table* yang terdapat pada *OpenFlow switch*. Jika sudah terdapat *flow* maka aplikasi akan melakukan *overwrite flow* dengan *flow* saat ini. Namun apabila belum terdapat *flow*, maka aplikasi akan membuat sebuah *flow* baru.

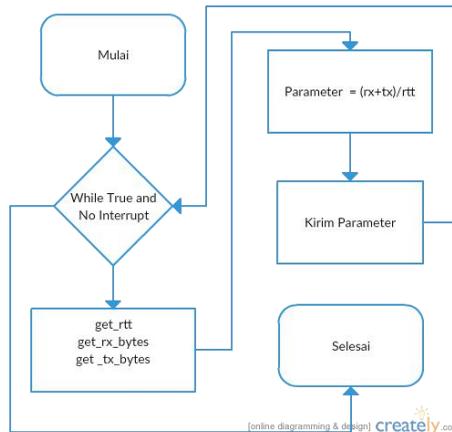
Alur sistem yang berjalan pada aplikasi *load switching* ini akan berjalan secara terus menerus selagi tidak terdapat *interrupt* dan *controller* masih berjalan dengan normal. Apabila terdapat *interrupt*, maka aplikasi akan berhenti dan harus menjalankan dari awal. Akan tetapi, data sebelumnya tetap dapat tersimpan dengan baik pada basis data *influxdb*. Diagram alir *load switching* ditunjukkan pada gambar 3.10 berikut ini.



Gambar 3.10. Diagram Alir Load Balancer

3.4.5. Diagram Alir Pengambilan Parameter

Proses pengambilan parameter pada *host router* ini dilakukan setelah proses *routing* dan *forwarding* selesai dijalankan. Proses ini melakukan penghitungan parameter menggunakan *throughput transfer (tx)*, *throughput receive (rx)*, dan *round trip time (rtt)*. Penghitungan hasil parameter dihitung berdasarkan rumus 3.3. Setelah parameter didapatkan, maka parameter dikirimkan ke *controller* untuk memodifikasi tabel *port* pada *controller*. Proses ini akan dijalankan terus menerus selama program tidak mendapat *interrupt* dan sistem tetap berjalan. Diagram alir pengambilan parameter ditunjukkan pada gambar 3.11 berikut ini.



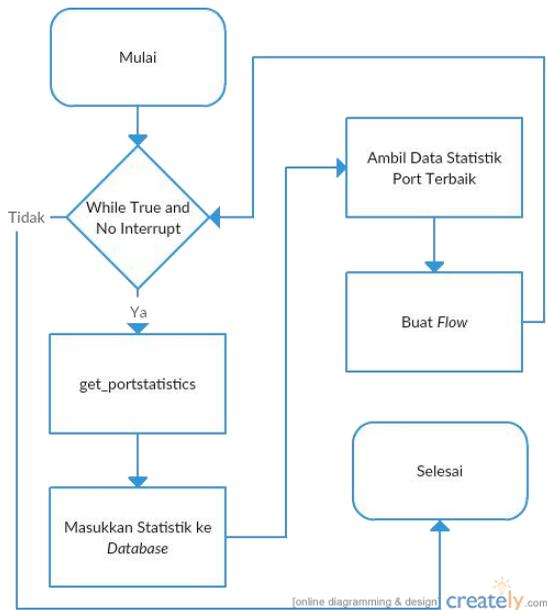
Gambar 3.11. Diagram Alir Pengambilan Parameter

3.4.6. Diagram Alir Penentuan Flow

Proses penentuan *flow* merupakan sebuah fungsi yang terdapat pada program *load switching*. Proses pembuatan *flow* yaitu sebuah proses untuk memberikan sebuah aturan (*policy*) terhadap suatu paket yang masuk. Aturan pada *flow* yang telah ditentukan ini kemudian dipasangkan pada perangkat *OpenFlow switch*. Oleh

karena itu, data *switch* ID sangat dibutuhkan untuk membuat *flow* dalam *Software Defined Network (SDN)*.

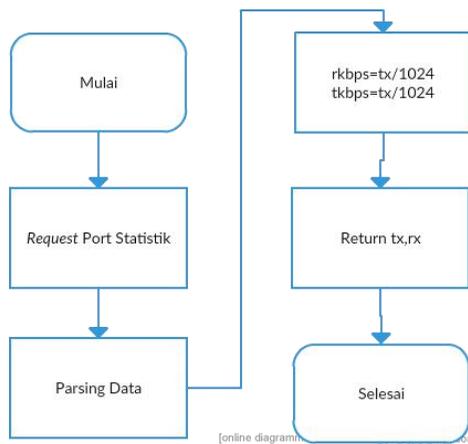
Proses penentuan *flow* dimulai dengan mengambil data statistik *port* yang terdapat pada OpenFlow *switch* dengan menggunakan REST API. Data tersebut kemudian dilakukan *parsing* yang kemudian dimasukkan kedalam basis data influxdb. Setelah data masuk kedalam basis data, maka program akan melakukan *query* data untuk mengambil data statistik dengan *port* yang terbaik (paling tinggi). Kemudian berdasarkan data hasil *query* tersebut maka program akan membuat sebuah *flow* yang dipasang ke OpenFlow *switch*. Proses ini dilakukan secara berulang dan terus menerus selama program tidak terdapat *interrupt*. Diagram alir penentuan *flow* ditunjukkan pada gambar 3.12 berikut ini.



Gambar 3.12. Diagram Alir Penentuan *Flow*

3.4.7. Diagram Alir Pengambilan Statistik *Throughput Controller*

Pengambilan statistik *throughput* pada *controller* merupakan sebuah fungsi yang terdapat pada aplikasi *load balancing* dan sangat dibutuhkan oleh fungsi penentuan *flow*. Proses pengambilan ini mengambil data *throughput* pada *controller* yang meliputi *tx_bytes*, dan *rx_bytes*. Kedua data tersebut diambil secara periodik setiap 1 (satu) detik kemudian dilakukan *parsing* data dan penghitungan *tx_bytes*, *rx_bytes* dalam satuan kbps. Setelah perhitungan selesai, maka hasil penghitungan *tx*, dan *rx* dikembalikan kepada fungsi yang memanggil proses ini. Diagram alir proses pengambilan *throughput controller* ditunjukkan pada gambar 3.13 berikut ini.

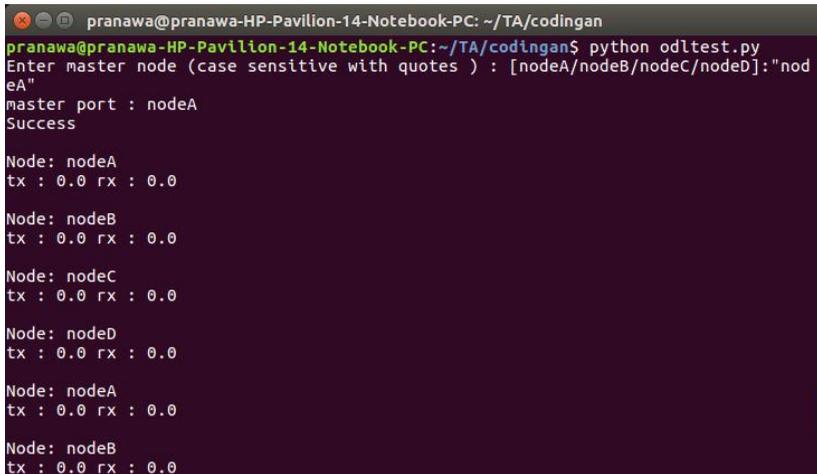


Gambar 3.13. Diagram Alir Pengambilan Statistik *Throughput Controller*

3.5. Perancangan Antarmuka *Logging* Sistem

Untuk dapat memantau apakah program dapat berjalan dengan baik, maka dibutuhkan sebuah mekanisme *logging* yang menampilkan informasi mengenai program yang sedang berjalan. Supaya informasi tersebut dapat mudah dimengerti, maka dibutuhkan

perancangan antarmuka yang jelas. Pada sistem ini, antarmuka *logging* sistem dibuat untuk menampilkan perubahan yang terjadi ketika program *load switching* dijalankan. Sistem antarmuka diimplementasikan dalam *Command Line Interface* (CLI) agar lebih cepat diproses oleh prosesor. Gambar 3.14 berikut ini merupakan *screenshot* dari rancangan antarmuka *logging* sistem.



```
pranawa@pranawa-HP-Pavilion-14-Notebook-PC: ~/TA/codingan
pranawa@pranawa-HP-Pavilion-14-Notebook-PC:~/TA/codingan$ python odltest.py
Enter master node (case sensitive with quotes) : [nodeA/nodeB/nodeC/nodeD]:"nodeA"
master port : nodeA
Success

Node: nodeA
tx : 0.0 rx : 0.0

Node: nodeB
tx : 0.0 rx : 0.0

Node: nodeC
tx : 0.0 rx : 0.0

Node: nodeD
tx : 0.0 rx : 0.0

Node: nodeA
tx : 0.0 rx : 0.0

Node: nodeB
tx : 0.0 rx : 0.0
```

Gambar 3.14. Tampilan Antarmuka *Logging* Sistem

[Halaman ini sengaja dikosongkan]

BAB IV IMPLEMENTASI

Bab ini membahas tentang proses implementasi pada sistem. Tahapan implementasi meliputi penentuan lingkungan implementasi, implementasi desain arsitektur jaringan, implementasi program dalam *pseudocode*, implementasi *traffic monitoring* pada antarmuka *web*.

4.1. Lingkungan Implementasi

Dalam proses implementasi sistem ini, digunakan beberapa perangkat pendukung untuk memenuhi kebutuhan sistem yang terdiri dari perangkat keras dan perangkat lunak.

4.1.1. Lingkungan Implementasi Perangkat Keras

Dalam proses implementasi, digunakan beberapa perangkat keras yang digunakan untuk memenuhi kebutuhan sistem, yaitu:

- 1 buah laptop Intel® Core™ i5-4200U CPU @1.6 GHz 64-bit dengan 4GB RAM sebagai SDN *controller*.
- 1 buah Mikrotik 8-port CRS109-8G-1S-2HnD-IN CPU @ 600 Mhz dengan 128MB RAM sebagai OpenFlow *switch*
- 4 buah modem 4G LTE tipe ZTE MF821 dengan masing-masing menggunakan provider Telkomsel, Indosat Ooredoo, XL, dan 3.
- 4 buah Raspberrypi tipe B Low Power ARM1176JZ-F @ 700 MHz dengan 512MB RAM sebagai *host router*.

4.1.2. Lingkungan Implementasi Perangkat Lunak

Sedangkan perangkat lunak yang digunakan untuk kebutuhan implementasi sistem diantaranya adalah:

- Linux Ubuntu 16.04 LTS 64-bit sebagai sistem operasi pada *controller*.
- RouterOS 6.37.3 dengan lisensi level 5 sebagai sistem operasi pada Mikrotik *router*.
- Raspbian Jessie sebagai sistem operasi pada *host router*.
- OpenDaylight Helium-SR4 OSGi *package* sebagai SDN *controller*.
- PyCharm community 2016.3 sebagai pengembangan python yang akan digunakan pada *controller* dan *host router*.
- Java OpenJDK 8 sebagai Java *environment* yang digunakan untuk menjalankan *controller*.
- Aplikasi wvdial sebagai *dialer* modem pada *host router*.
- Postman sebagai aplikasi untuk melakukan pengujian fungsionalitas REST API
- Influxdb sebagai aplikasi *database* untuk menyimpan data *throughput stream* yang dikirim oleh *host router*.
- Grafana sebagai aplikasi untuk merepresentasikan *User Interface* dari *database* influxdb.
- WPS Office writer sebagai aplikasi pengolah kata untuk pembuatan dokumentasi dan penyelesaian buku tugas akhir.

4.2. Implementasi Perangkat Lunak

Implementasi perangkat lunak terbagi menjadi tiga, yaitu implementasi pada *controller*, implementasi pada *host router*, dan implementasi pada *router*. Implementasi pada *controller* merupakan implementasi untuk pembuatan aplikasi *load switching* dan implementasi sistem *controller*. Implementasi pada *router* merupakan implementasi untuk konfigurasi protokol OpenFlow. Implementasi pada *host router* merupakan implementasi untuk *dialer modem*, *forwarding*, dan deteksi *throughput*.

4.2.1. Implementasi pada *Controller*

Pada subbab-subbab berikut ini akan dijelaskan mengenai implementasi pada *controller*. Penjelasan implementasi akan ditulis mengenai instalasi program yang dibutuhkan, inisialisasi *controller*, dan penjelasan kode program *load switching* menggunakan *pseudocode*.

4.2.1.1. Instalasi *Controller*

Proses instalasi program pada *controller* ini diterapkan pada sebuah laptop. Tahap awal dilakukan proses pemasangan java openjdk 8 kemudian dilanjutkan dengan melakukan pengunduhan dari *repository* OpenDaylight. Sebagai catatan, *controller* OpenDaylight yang digunakan ini adalah OpenDaylight helium SR4 yang hanya di support dengan baik oleh openjdk 7 dan openjdk 8. Proses instalasi *controller* dapat dilihat pada gambar 4.1 berikut ini.

```
$ sudo apt-get install openjdk-8-jre  
  
$ wget  
https://nexus.opendaylight.org/content/repositories/OpenDaylight.release/org/OpenDaylight/controller/distribution.opendaylight/0.1.6
```

```
-Helium-SR4/distribution.OpenDaylight-0.1.6-
Helium-SR4-osgipackage.zip

$ unzip distribution.OpenDaylight-0.1.6-
Helium-SR4-osgipackage.zip
```

Gambar 4.1. Proses Inisialisasi *Controller*

4.2.1.2. Proses Inisialisasi *Controller*

Proses inisialisasi *controller* dimulai dengan melakukan setting pada *java environment* agar dapat digunakan oleh *controller*. Perintah yang dituliskan dalam terminal linux adalah sebagai berikut

```
$ export JAVA_HOME=/usr/lib/jvm/java-xx-
openjdk-yy
```

Gambar 4.2. Perintah Pengaturan *Environment*

xx menunjukkan versi *openjdk* yang digunakan, sedangkan yy menunjukkan versi arsitektur komputer yang digunakan, 32-bit atau 64-bit.

Untuk menjalankan aplikasi *controller*, maka perintah yang diketikkan pada terminal linux adalah sebagai berikut

```
$ sudo ./run.sh
```

Gambar 4.3. Perintah Menjalankan Aplikasi *Controller*

4.2.1.3. Konfigurasi *Controller*

Pada saat pertama kali *controller* dijalankan, *controller* belum sepenuhnya dapat digunakan. Oleh karena itu, dibutuhkan sebuah konfigurasi agar *controller* yang dijalankan dapat dapat berfungsi dengan arsitektur jaringan yang sedang digunakan.

Untuk melakukan konfigurasi *controller* maka dilakukan melalui antarmuka *web* pada port 8080 seperti pada gambar 4.4 berikut ini.

```
http://localhost:8080/
```

Gambar 4.4. Alamat Antarmuka *Controller*

4.2.1.4. Proses Inisialisasi *load switching*

Proses inisialisasi aplikasi *load switching* adalah proses menjalankan aplikasi *load switching* yang berjalan pada komputer yang bertindak sebagai *controller*. Proses ini hanya dijalankan satu kali ketika sistem berjalan. Proses ini akan menggunakan REST API yang disediakan oleh *controller* untuk berkomunikasi dengan *controller*. Implementasi aplikasi ini menggunakan bahasa *python* dengan menggunakan *library sys, os, httplib2, time, dan influxdb*. Perintah pada terminal linux untuk menjalankan aplikasi *load switching* ditunjukkan pada gambar 4.5 berikut ini.

```
$ python odlloadbalancing.py
```

Gambar 4.5. Perintah Menjalankan Inisialisasi *load switching*

Pseudocode aplikasi ini ditunjukkan pada gambar 4.6 berikut ini.

```
1. portstatistics=rx_bytes
2. best_port,last_best_port=masterport
3. flowexist=FALSE
4. while (TRUE)
5. portstatistics=get_port_statistics()
6. insert_to_database(portstatistics)
7. best_port = 'SELECT max(param) FROM
   port_table' WHERE time>(now()-xs)
8. if (best_port>last_best_port)
```

```

9. flowexist = check_flow_table()
10. if (flowexist == TRUE)
11. overwrite_flow()
12. else
13. write_new_flow()

```

Gambar 4.6. Pseudocode Inisialisasi load switching

4.2.1.5. Proses Pengambilan Statistik *Throughput*

Proses pengambilan data statistik *throughput* ini merupakan suatu fungsi yang terdapat pada aplikasi *load switching*. Ketika aplikasi *load switching* berjalan, maka proses pengambilan data statistik *throughput* akan melakukan pengecekan tabel *throughput* pada *controller* setiap satu detik. *Pseudocode* proses ini ditunjukkan pada gambar 4.7 berikut.

```

1. function get_port_statistics():
2. portstats=request_port_stats_API()
3. portdata=json_parsing(portstats)
4. rx_kbps=portdata[rx]/1024
5. tx_kbps=portdata[tx]/1024
6. return rx_kbps,tx_kbps
7. end function

```

Gambar 4.7. Pseudocode Pengambilan Statistik *Throughput*

4.2.1.6. Proses Memasukkan Data ke *Database*

Proses memasukkan data ke *database* ini dilakukan setelah proses pengambilan statistik *throughput*. Fungsi ini akan menerima data *throughput* kemudian memasukkan data tersebut ke *database influxdb*. *Pseudocode* pada proses ini ditunjukkan pada gambar 4.8 berikut.

```

1. function insert_database():
2. port_data=get_stats()
3. query='INSERT INTO stats,port_data'
4. return query
5. end function

```

Gambar 4.8. Pseudocode Memasukkan Data ke Database

4.2.1.7. Proses Penentuan *Flow*

Proses penentuan *flow* merupakan proses untuk memberikan *policy* pada suatu paket dan melakukan *forwarding* ke suatu alamat tertentu. Penentuan *flow* didasari pada *packet match* pada suatu paket yang masuk kedalam suatu *port*. Kemudian paket tersebut dimodifikasi dengan menambahkan suatu *action* tertentu. Pada aplikasi ini, penentuan *flow* dimodifikasi dengan menambahkan parameter berdasarkan *throughput* terbaik yang diperoleh dari hasil pengambilan statistik *throughput*. *Pseudocode* proses penentuan *flow* ditunjukkan pada gambar 4.9 berikut ini.

```

1. function write_flow():
2. while (TRUE)
3. portstats=get_port_statistics()
4. insert_database()
5. best_port = 'SELECT max(param) FROM
               port_table' WHERE time>(now()-xs)
6. write_flow()
7. end function

```

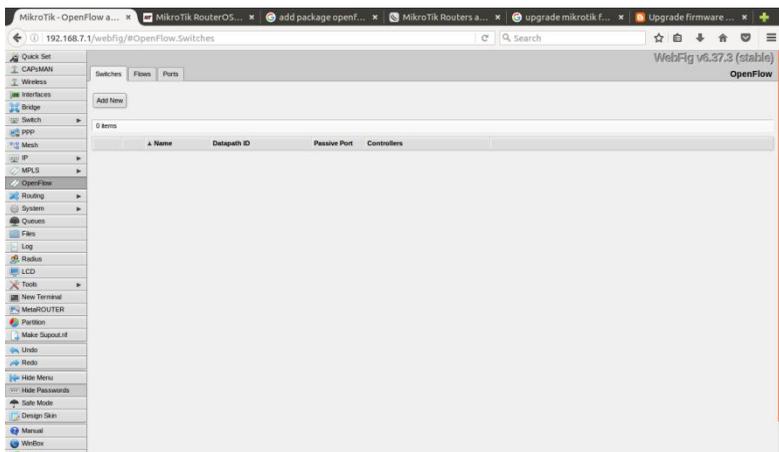
Gambar 4.9. Pseudocode Penentuan *Flow*

4.2.2. Implementasi pada *Router*

Implementasi pada *router* terbagi menjadi dua proses, yaitu instalasi fitur OpenFlow dan konfigurasi protokol OpenFlow. Pada implementasi ini digunakan *router* mikrotik *cloud router series* dengan routerOS 6.37.3 dan lisensi level 5.

4.2.2.1. Instalasi Fitur OpenFlow

Pada dasarnya semua *router* mikrotik sudah mendukung fitur OpenFlow versi 1.0. Akan tetapi terdapat beberapa tipe *router* mikrotik yang tidak memiliki fitur OpenFlow secara langsung. Fitur OpenFlow tersebut harus dilakukan instalasi menggunakan sebuah paket yang terpisah. Berikut ini adalah tampilan antarmuka sebuah *router* telah terpasang protokol OpenFlow.



Gambar 4.10. Antarmuka *Router* dengan Protokol OpenFlow

4.2.2.2. Konfigurasi OpenFlow

Konfigurasi protokol OpenFlow dilakukan setelah fitur OpenFlow terpasang pada sistem *router* mikrotik. Protokol

OpenFlow pada *router* mikrotik digunakan sebagai standar komunikasi yang dilakukan antara *router* dengan *controller*. Setelah konfigurasi fitur OpenFlow dilakukan dan *router* dapat berkomunikasi dengan *controller*, maka semua aturan yang terdapat pada *router* meliputi *routing*, *forwarding*, dan sebagainya diatur menggunakan *controller*. Sehingga konfigurasi yang terdapat pada *routerOS* mikrotik tidak digunakan lagi. Agar sebuah *port* dapat digunakan sebagai *port enabled* OpenFlow, maka *port* tersebut harus dikonfigurasi sebagai *master-port*. Gambar 4.11 berikut ini menjelaskan cara melakukan konfigurasi OpenFlow.

```
[admin@MikroTik] > interface ethernet print
Flags: X - disabled, R - running, S - slave
#  NAME      MTU  MAC-ADDRESS  ARP      MASTER-PORT  SWITCH
0  ether1    1500 E4:8D:8C:AF:43:2E enabled    none          switch1
1  S ether2... 1500 E4:8D:8C:AF:43:2F enabled    none          switch1
2  S ether3    1500 E4:8D:8C:AF:43:30 enabled    ether2-master switch1
3  S ether4    1500 E4:8D:8C:AF:43:31 enabled    ether2-master switch1
4  S ether5    1500 E4:8D:8C:AF:43:32 enabled    ether2-master switch1
5  S ether6    1500 E4:8D:8C:AF:43:33 enabled    ether2-master switch1
6  S ether7    1500 E4:8D:8C:AF:43:34 enabled    ether2-master switch1
7  S ether8    1500 E4:8D:8C:AF:43:35 enabled    ether2-master switch1
8  S sfp1     1500 E4:8D:8C:AF:43:36 enabled    ether2-master switch1

[admin@MikroTik] /interface ethernet> set ether3,ether4,ether5,ether6 master-port=none
[admin@MikroTik] /interface ethernet> print
Flags: X - disabled, R - running, S - slave
#  NAME      MTU  MAC-ADDRESS  ARP      MASTER-PORT  SWITCH
0  ether1    1500 E4:8D:8C:AF:43:2E enabled    none          switch1
1  S ether2... 1500 E4:8D:8C:AF:43:2F enabled    none          switch1
2  ether3    1500 E4:8D:8C:AF:43:30 enabled    none          switch1
3  ether4    1500 E4:8D:8C:AF:43:31 enabled    none          switch1
4  ether5    1500 E4:8D:8C:AF:43:32 enabled    none          switch1
5  ether6    1500 E4:8D:8C:AF:43:33 enabled    none          switch1
6  S ether7    1500 E4:8D:8C:AF:43:34 enabled    ether2-master switch1
7  S ether8    1500 E4:8D:8C:AF:43:35 enabled    ether2-master switch1
8  S sfp1     1500 E4:8D:8C:AF:43:36 enabled    ether2-master switch1
```

```
[admin@MikroTik] > OpenFlow add
name=ofswitch1 controllers=192.168.88.254
[admin@MikroTik] > OpenFlow enable ofswitch1
[admin@MikroTik] > OpenFlow port add
```

```

switch=ofswitch1 interface=ether3
[admin@MikroTik] > OpenFlow port add
switch=ofswitch1 interface=ether4
[admin@MikroTik] > OpenFlow port add
switch=ofswitch1 interface=ether5
[admin@MikroTik] > OpenFlow port add
switch=ofswitch1 interface=ether6
[admin@MikroTik] > OpenFlow port enable
numbers=0
[admin@MikroTik] > OpenFlow port enable
numbers=1
[admin@MikroTik] > OpenFlow port enable
numbers=2
[admin@MikroTik] > OpenFlow port enable
numbers=3

```

Gambar 4.11. Konfigurasi OpenFlow pada Router

4.2.3. Implementasi *Host Router*

Implementasi *host router* merupakan sebuah proses untuk mengubah fungsi *host* menjadi sebuah *router* yang terhubung dengan *internet*. Proses implementasi pada *host router* ini dibagi menjadi tiga proses, yaitu proses *dial-up*, proses *forwarding*, dan proses pengambilan dan pengiriman data *throughput* ke *controller*. Implementasi pada *host router* ini menggunakan komputer Raspberrypi tipe B dan menggunakan sistem operasi raspbian.

4.2.3.1. Proses *Dial-Up*

Proses *dial-up* pada *host router* menggunakan sebuah *usb modem* dan menggunakan 4 operator seluler, yaitu Telkomsel, Indosat Ooredoo, Three, dan XL. Masing-masing operator terhubung dengan satu *host router*. Untuk melakukan proses *dial-up* maka dibutuhkan instalasi aplikasi *wvdial* dan melakukan konfigurasi seperti pada gambar 4.12 berikut ini.

```

$ sudo apt-get update

$ sudo apt-get install ppp usb-modeswitch
wvdial

$ sudo wvdialconf

$ sudo wvdial

```

Gambar 4.12. Proses Instalasi Aplikasi *Dial-up*

4.2.3.2. Proses *Forwarding*

Proses *forwarding* merupakan sebuah proses yang dilakukan untuk meneruskan sebuah paket yang masuk menuju jaringan *internet* melalui sebuah *interface* jaringan. Pada sistem ini akan melakukan proses *forwarding* paket yang masuk melalui *interface* *eth0* menuju ke *internet* melalui *interface* *ppp0*. Konfigurasi pada proses *forwarding* ini ditunjukkan melalui gambar 4.13 berikut ini.

```

$ sudo iptables -t nat -A POSTROUTING -o
ppp0 -j MASQUERADE

$ sudo iptables -A FORWARD -i ppp0 -o eth0 -
m state --state RELATED,ESTABLISHED -j
ACCEPT

$ sudo iptables -A FORWARD -i eth0 -o ppp0 -
j ACCEPT

```

Gambar 4.13. Konfigurasi Proses *Forwarding Host Router*

4.2.3.3. Proses Pengambilan dan Pengiriman *Throughput*

Proses pengambilan dan pengiriman *throughput* pada *host router* dilakukan dengan melakukan monitoring *throughput* pada

interface yang terhubung dengan *internet* (dalam sistem ini *interface* yang digunakan adalah *ppp0*). Setelah melakukan monitoring, maka program akan mencatat perubahan *throughput* setiap detik. Kemudian program akan mengirimkan data *throughput* tersebut kepada *controller* melalui protokol ICMP dengan paket sebesar data *throughput*. *Pseudocode* proses pengambilan dan pengiriman data *throughput* dapat dilihat pada gambar 4.14 berikut ini.

```
1. function get_parameter():
2. while (TRUE)
3. ping_ip_address()
4. round_trip_time=get_rtt()
5. tx_bytes=get_tx()
6. rx_bytes=get_rx()
7. parameter=(tx_bytes+rx_bytes)/round_trip_
   time
8. send_parameter(parameter)
9. end function
```

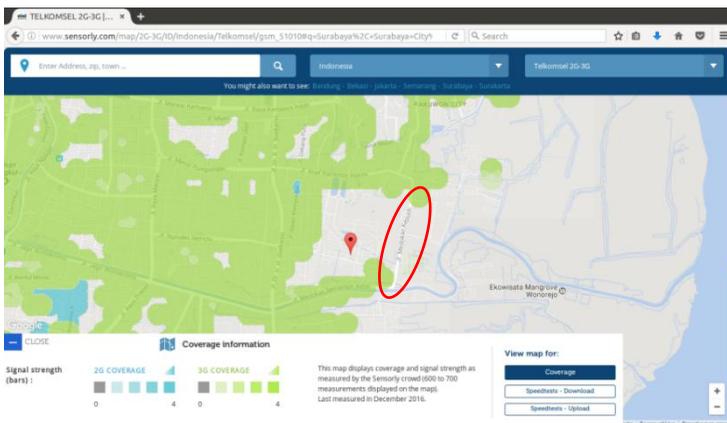
Gambar 4.14. *Pseudocode* Proses Pengambilan dan Pengiriman *Throughput*

4.3. Implementasi Rute

Implementasi rute memiliki tujuan untuk menentukan daerah yang memiliki kualitas jaringan yang kurang stabil. Rute dipilih berdasarkan analisis *coverage*, analisis *download*, dan analisis *upload* pada suatu wilayah. Aplikasi yang digunakan sebagai acuan analisis adalah www.sensorly.com yang menyediakan data statistik jaringan seluler dengan menggunakan antarmuka peta.

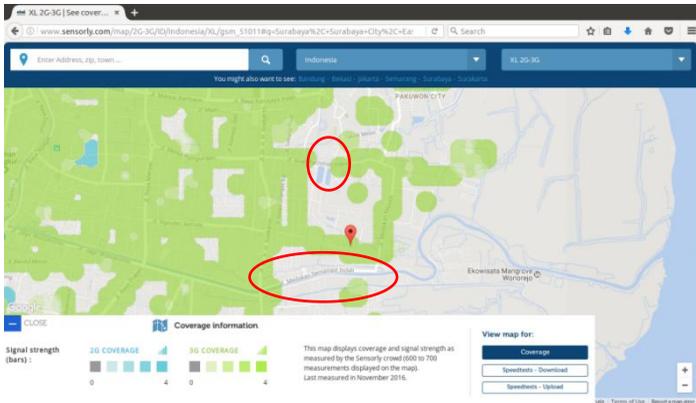
4.3.1. Analisis Coverage

Coverage merupakan sebuah wilayah yang dapat dijangkau oleh penyedia jaringan tertentu. *Coverage* pada pengguna direpresentasikan dengan kekuatan sinyal yang dapat berupa garis (*bar*) maupun angka. Pada gambar 4.15 sampai 4.18 dibawah ini menunjukkan sebuah *coverage area* masing-masing penyedia jaringan (telkomsel, Indosat Ooredoo, XL, dan Three). Data pada sensorly.com menunjukkan bahwa terdapat perbedaan *coverage area* yang berlokasi di daerah Keputih, Surabaya. Hal ini utamanya ditunjukkan pada jalan Medokan Keputih, jalan Medokan Semampir Indah, dan jalan Arief Rachman Hakim. Pada ketiga jalan tersebut, representasi kekuatan sinyal menunjukkan perbedaan yang cukup signifikan.



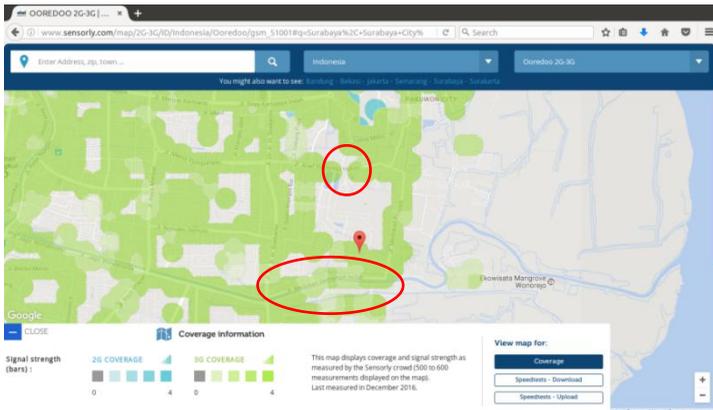
Gambar 4.15. Coverage Area menggunakan Provider Telkomsel

Gambar 4.15 diatas adalah gambar *coverage area* yang dapat dicakup oleh *provider* Telkomsel. Data tersebut merupakan data yang diambil dari sensorly.com. Dapat dilihat bahwa terdapat daerah yang tidak dapat terjangkau oleh *provider* Telkomsel, yaitu di jalan Medokan Keputih.



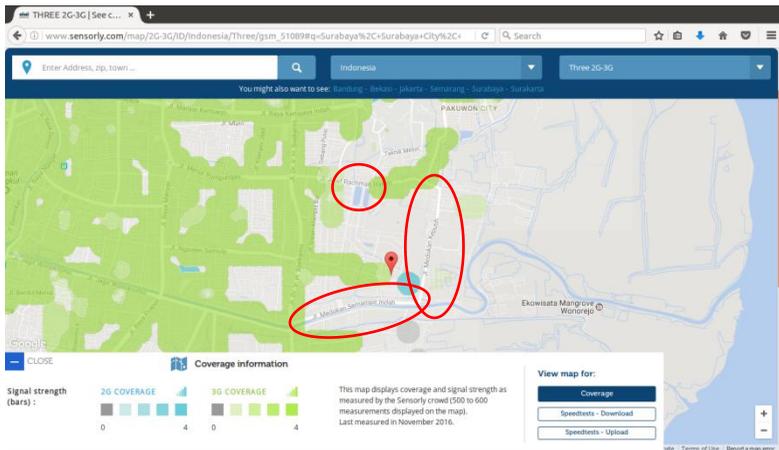
Gambar 4.16. Coverage Area menggunakan Provider XL

Gambar 4.16 diatas adalah gambar *coverage area* yang dapat dicakup oleh *provider* XL. Data tersebut merupakan data yang diambil dari *sensorly.com*. Dapat dilihat bahwa terdapat daerah yang tidak dapat terjangkau oleh *provider* XL, yaitu di jalan Medokan Semampir Indah dan sebagian kecil di jalan Arief Rachman Hakim.



Gambar 4.17. Coverage Area menggunakan Provider Indosat Ooredoo

Gambar 4.17 diatas adalah gambar *coverage area* yang dapat dicakup oleh *provider* Indosat Ooredoo. Data tersebut merupakan data yang diambil dari sensorly.com. Dapat dilihat bahwa terdapat daerah yang tidak dapat terjangkau oleh *provider* Indosat Ooredoo, yaitu sebagian kecil di jalan Arief Rachman Hakim dan terdapat sinyal yang sangat rendah di jalan Medokan Semampir Indah.



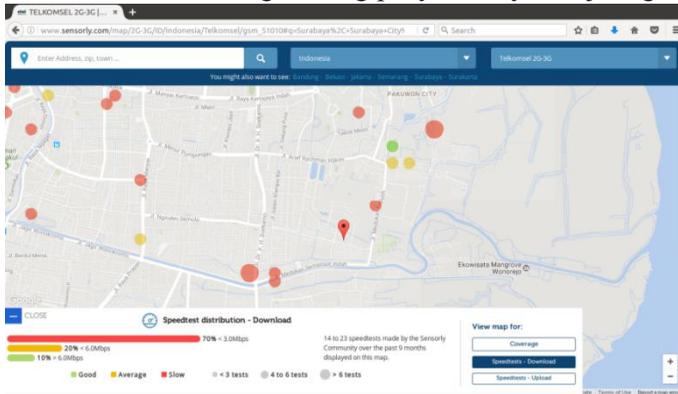
Gambar 4.18. Coverage Area menggunakan Provider Three

Gambar 4.18 diatas adalah gambar *coverage area* yang dapat dicakup oleh *provider* Three. Data tersebut merupakan data yang diambil dari sensorly.com. Dapat dilihat bahwa terdapat daerah yang tidak dapat terjangkau oleh *provider* Three, yaitu di jalan Medokan Keputih, jalan Medokan Semampir Indah, dan sebagian kecil di jalan Arief Rachman Hakim.

4.3.2. Analisis *Download*

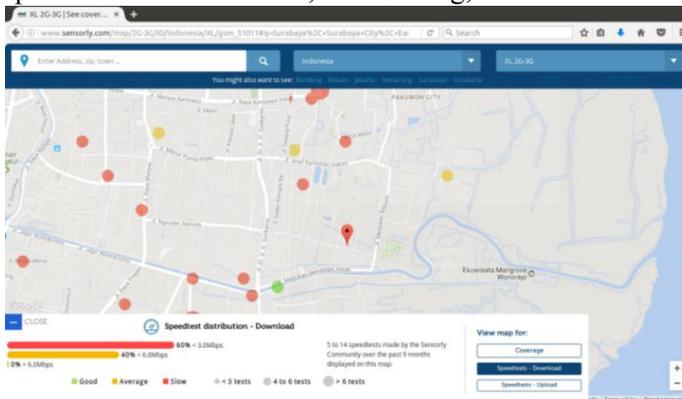
Download speed atau kecepatan unduh merupakan sebuah proses pertukaran data yang mana suatu perangkat melakukan pengambilan data dari perangkat lain. *Download speed* menjadi sebuah parameter penting dalam menentukan kualitas sebuah

jaringan karena semua perangkat yang berkomunikasi melalui jaringan pasti melakukan proses pengambilan data atau *request*. Pada gambar 4.19 sampai 4.22 berikut ini menunjukkan data statistik unduh dari masing-masing penyedia layanan jaringan.



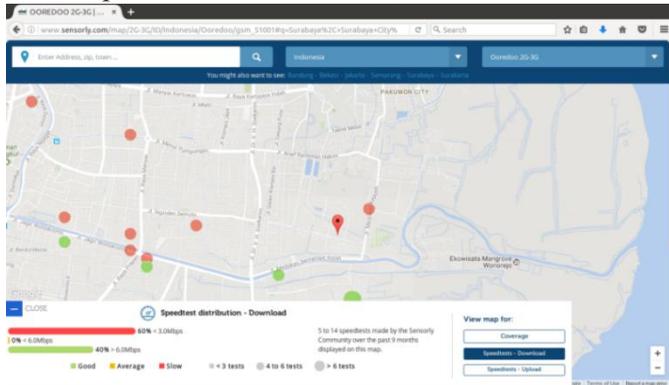
Gambar 4.19. Representasi Kecepatan Unduh *Provider* Telkomsel

Gambar 4.19 diatas merupakan gambar representasi kecepatan unduh dari *provider* Telkomsel. Data tersebut merupakan data yang diambil dari website sensorly.com. Dapat dilihat bahwa pada daerah ini telkomsel memiliki statistik kecepatan unduh 70% lambat, 20% sedang, dan 10% baik.



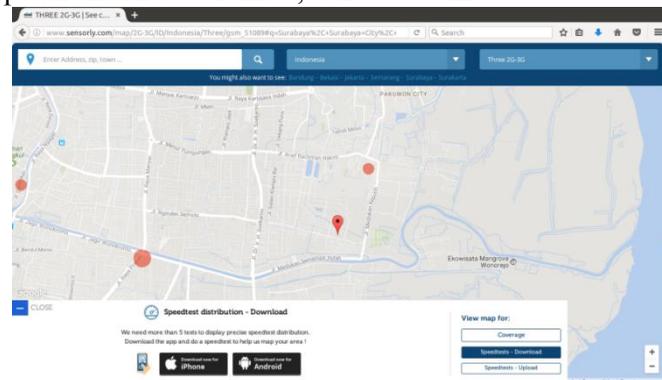
Gambar 4.20. Representasi Kecepatan Unduh *Provider* XL

Gambar 4.20 diatas merupakan gambar representasi kecepatan unduh dari *provider* XL. Berdasarkan data sensorly.com dapat dilihat bahwa pada daerah ini XL memiliki statistik kecepatan unduh 60% lambat, dan 40% baik.



Gambar 4.21. Representasi Kecepatan Unduh *Provider* Indosat Ooredoo

Gambar 4.21 diatas merupakan gambar representasi kecepatan unduh dari *provider* Indosat Ooredoo. Data tersebut merupakan data yang diambil dari website sensorly.com. Dapat dilihat bahwa pada daerah ini Indosat Ooredoo memiliki statistik kecepatan unduh 60% lambat, dan 40% baik.

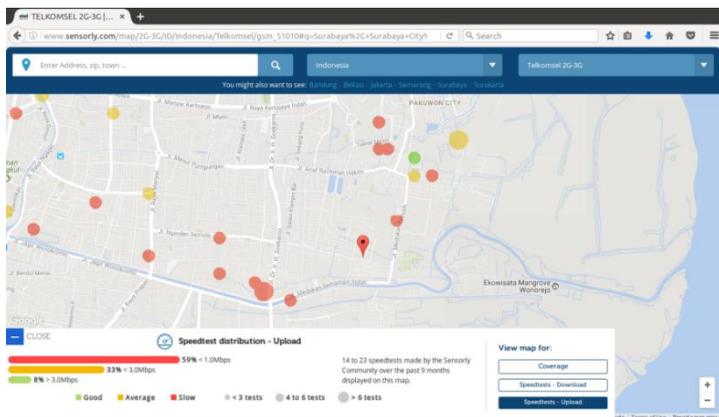


Gambar 4.22. Representasi Kecepatan Unduh *Provider* Three

Gambar 4.22 diatas merupakan gambar representasi kecepatan unduh dari *provider* Three. Data tersebut merupakan data yang diambil dari website sensorly.com. Dapat dilihat bahwa website sensorly.com belum memiliki data yang cukup untuk merepresentasikan kecepatan unduh dari *provider* Three pada daerah ini.

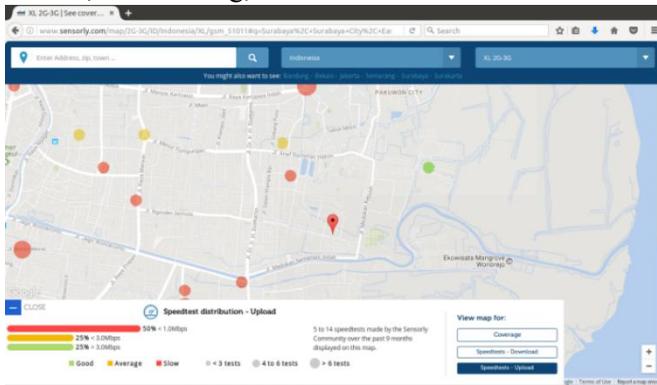
4.3.3. Analisis *Upload*

Hampir sama seperti data *download*, data *upload* atau data unggah yang dianalisis ini merupakan sebuah data penting yang dapat dijadikan parameter penentu suatu kualitas jaringan. Pada mekanisme *upload*, suatu perangkat melakukan proses pengiriman data kepada perangkat lain dalam jaringan. Mekanisme ini juga pasti dilakukan oleh suatu perangkat dalam jaringan, karena tujuan dari suatu perangkat terhubung pada jaringan adalah untuk pertukaran informasi. Pada gambar 4.23 sampai 4.26 berikut ini menunjukkan representasi data *upload* pada wilayah yang sama dengan wilayah *coverage area*. Data *upload* yang ditampilkan merupakan data *upload* dari beberapa penyedia layanan jaringan.



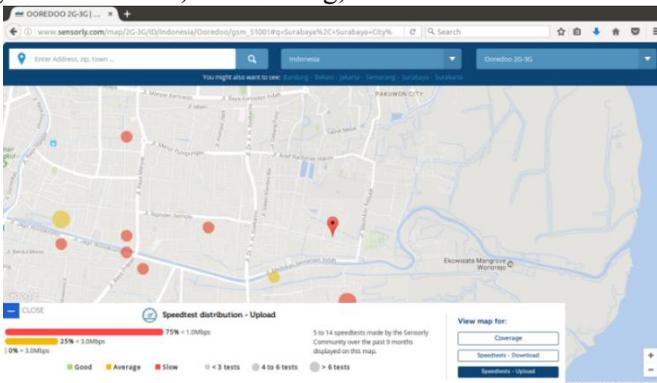
Gambar 4.23. Representasi Kecepatan Unggah *Provider* Telkomsel

Gambar 4.23 diatas merupakan representasi unggah menggunakan *provider* Telkomsel. Data tersebut diambil berdasarkan dara dari sensorly.com. Dapat dilihat bahwa pada daerah tersebut *provider* Telkomsel memiliki statistik unggah 59% lambat, 33% sedang, dan 8% baik.



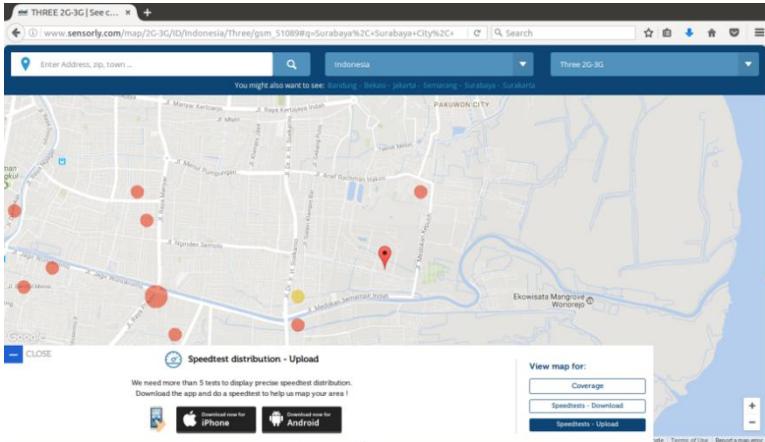
Gambar 4.24. Representasi Kecepatan Unggah *Provider* XL

Gambar 4.24 diatas merupakan representasi unggah menggunakan *provider* XL. Berdasarkan data sensorly.com dapat dilihat bahwa pada daerah tersebut *provider* XL memiliki statistik unggah 50% lambat, 25% sedang, dan 25% baik.



Gambar 4.25. Representasi Kecepatan Unggah *Provider* Indosat Ooredoo

Gambar 4.25 diatas merupakan representasi unggah menggunakan *provider* Indosat Ooredoo. Data tersebut diambil berdasarkan dara dari sensorly.com. Dapat dilihat bahwa pada daerah tersebut *provider* Indosat Ooredoo memiliki statistik unggah 75% lambat dan 25% sedang.

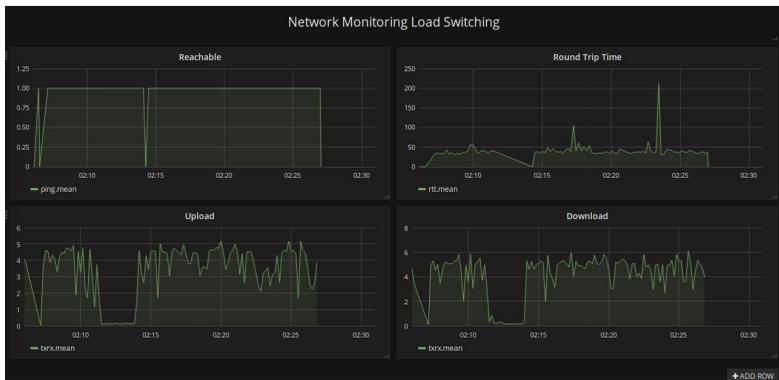


Gambar 4.26. Representasi Kecepatan Unggah *Provider* Three

Gambar 4.26 diatas merupakan representasi unggah menggunakan *provider* Three. Data tersebut diambil berdasarkan dara dari sensorly.com. Dapat dilihat bahwa pada daerah tersebut data sensorly.com belum dapat menampilkan representasi data unggah yang cukup untuk *provider* Three.

4.4. Implementasi Antarmuka Perangkat Lunak

Implementasi antarmuka berupa tampilan grafik monitoring jaringan menggunakan Grafana. Grafik yang ditampilkan meliputi kondisi *reachable* suatu jaringan, *Round Trip Time (RTT)*, *throughput transfer bytes (tx)*, dan *throughput received bytes (rx)*. Gambar 4.27 berikut ini merupakan *screenshot* dari tampilan antarmuka sistem yang dibuat.



Gambar 4.27. Tampilan Antarmuka *Monitoring Grafana*

[Halaman ini sengaja dikosongkan]

BAB V

UJI COBA DAN EVALUASI

Pada bab ini dibahas mengenai uji coba dan evaluasi dari segi fungsionalitas dan performa dari sistem. Uji coba fungsionalitas dan performa dibagi ke dalam beberapa skenario uji coba fungsi-fungsi utama dalam sistem. Pengujian performa meliputi pengukuran perbandingan nilai *throughput* sebelum menggunakan aplikasi dengan nilai *throughput* setelah menggunakan aplikasi *load switching* pada *software defined network*.

5.1. Lingkungan Uji Coba

Dalam proses uji coba, digunakan beberapa perangkat keras, perangkat lunak, dan rute perjalanan sebagai penunjang kebutuhan pengoperasian yang akan dijelaskan pada subbab berikut.

5.1.1. Perangkat Keras

Uji coba sistem dilakukan dengan menggunakan 2 (dua) buah komputer sebagai *client* dan *controller*, 1 (satu) buah *router*, 4 (empat) buah *host router*, dan 4 (empat) buah *modem*. Perangkat tersebut kemudian diletakkan dan dijalankan di atas *dashboard* mobil seperti pada gambar 5.1 berikut ini. Untuk mengubah arus 12 volt dari aki menjadi arus 220 volt, dibutuhkan alat tambahan yaitu *inverter* yang mengubah arus DC menjadi arus AC. Daya yang dihasilkan oleh *inverter* ini sebesar 150 watt, sehingga cukup untuk menjalankan keseluruhan sistem secara bersama-sama.

Tabel 1. Perangkat Keras Uji Coba Sistem

Perangkat Keras (Laptop)	
Prosesor	Intel® Pentium 8980 CPU @2.4 GHz 64-bit

RAM	2GB RAM
GPU	Nvidia Geforce 610M 2GB
Sistem Operasi	Ubuntu 16.04 64-bit
Kegunaan	Sebagai <i>client</i>
Perangkat Keras (Laptop)	
Prosesor	Intel® Core™ i5-4200U CPU @1.6 GHz 64-bit
RAM	4GB RAM
GPU	Intel® Haswell Mobile
Sistem Operasi	Ubuntu 16.04 64-bit
Kegunaan	Sebagai <i>controller</i>
Perangkat Keras (Router)	
Tipe	Mikrotik 8-port CRS109-8G-1S-2HnD-IN
CPU	600 Mhz
RAM	128MB RAM
Port	8
Kegunaan	Sebagai OpenFlow <i>switch</i>
Perangkat Keras (Raspberrypi)	
Prosesor	ARM1176JZ-F @ 700 MHz
RAM	512MB RAM
GPU	Videocore 4 GPU
Sistem Operasi	Raspbian Jessie
Kegunaan	Sebagai <i>host router</i>
Perangkat Keras (Modem)	
Tipe	ZTE MF821
Kekuatan sinyal	LTE/WCDMA/GSM
Kegunaan	Sebagai <i>modem</i>



Gambar 5.1. Implementasi Uji Coba pada *Dashboard Mobil*

5.1.2. Perangkat Lunak

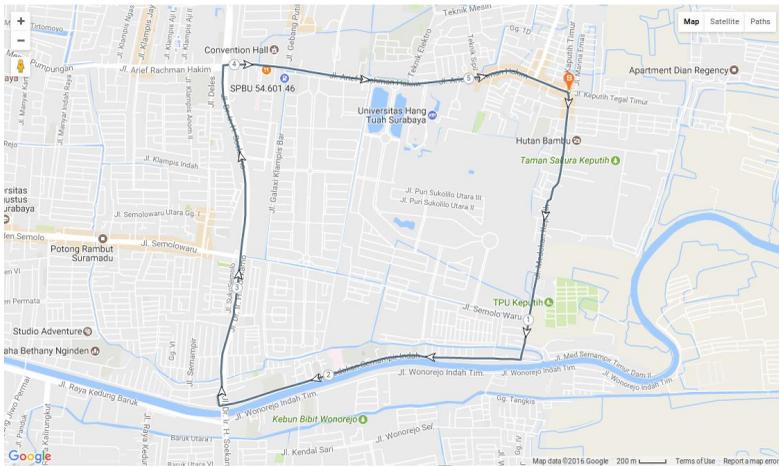
Kebutuhan perangkat lunak yang digunakan dalam uji coba sistem yaitu:

- Linux Ubuntu 16.04 LTS 64-bit sebagai sistem operasi pada *controller*.
- RouterOS 6.37.3 dengan lisensi level 5 sebagai sistem operasi pada Mikrotik *router*.
- Raspbian Jessie sebagai sistem operasi pada *host router*.
- OpenDaylight Helium-SR4 OSGi *package* sebagai SDN *controller*.
- PyCharm community 2016.3 sebagai pengembangan python yang akan digunakan pada *controller* dan *host router*.
- Java OpenJDK 8 sebagai Java *environment* yang digunakan untuk menjalankan *controller*.

- Aplikasi `wvdial` sebagai *dialer* modem pada *host router*.
- Postman sebagai aplikasi untuk melakukan pengujian fungsionalitas REST API
- Influxdb sebagai aplikasi *database* untuk menyimpan data *throughput stream* yang dikirim oleh *host router*.
- Grafana sebagai aplikasi untuk merepresentasikan *User Interface* dari *database influxdb*.

5.1.3. Rute Perjalanan

Rute perjalanan pada uji coba sistem *load switching* dengan *software defined network* ini ditentukan berdasarkan analisis rute pada BAB IV. Pada uji coba yang dijalankan, sistem akan dipasang pada mobil dengan menggunakan *inverter* daya yang mengubah arus DC 12 volt menjadi arus AC 220 volt dan daya 150 watt. Setelah sistem dipasang pada mobil, kemudian sistem dijalankan dan dilakukan uji coba dengan berkendara melalui rute yang telah ditentukan. Pada putaran pertama akan diuji coba untuk pengambilan data sebelum menggunakan *load switching*, putaran kedua akan menggunakan *load switching* tanpa waktu tunggu, putaran ketiga akan menggunakan *load switching* dengan waktu tunggu. Kemudian ketiga data sebelum menggunakan sistem *load switching* dan setelah menggunakan sistem *load switching* akan dibandingkan hasilnya. Gambar 5.2 berikut merepresentasikan rute yang telah ditentukan untuk melakukan uji coba.



Gambar 5.2. Rute yang Ditempuh untuk Uji Coba

5.2. Uji Coba Fungsionalitas

Uji coba fungsionalitas merupakan sebuah pengujian terhadap jalannya fungsi-fungsi utama yang ada pada aplikasi. Uji coba fungsionalitas meliputi:

1. Inisialisasi *controller*
2. Inisialisasi *load switching*
3. Pengambilan Statistik *throughput*
4. Memasukkan Data ke *database*
5. Penentuan *flow*
6. Proses *dial-up*
7. *Forwarding* paket
8. Pengambilan dan pengiriman *throughput*
9. Inisialisasi *Network Monitoring*

5.2.1. Inisialisasi *Controller*

Proses pertama yang dilakukan ketika sistem *load switching* jaringan seluler dengan *software defined network* adalah melakukan inisialisasi *controller*. Inisialisasi *controller* adalah sebuah proses untuk menjalankan aplikasi *controller software defined network* yaitu OpenDaylight Helium SR-4 OSGi *package*. Proses ini ditandai dengan keluarnya *log* pada *terminal* yang menampilkan notifikasi bahwa inisialisasi *controller* telah berhasil dimuat. Tabel 2 menunjukkan prosedur uji coba yang dilakukan pada proses inisialisasi *controller*.

Tabel 2. Prosedur Uji Coba Inisialisasi *Controller*

ID	UJ-01
Nama	Uji Coba Proses Inisialisasi <i>Controller</i>
Tujuan Uji Coba	Menguji proses inisialisasi <i>controller</i> OpenDaylight Helium SR-4 dengan OSGi <i>package</i> kedalam sistem operasi Ubuntu 16.04
Kondisi Awal	<i>Controller</i> belum berjalan
Skenario	Menjalankan aplikasi <i>controller</i> OpenDaylight Helium SR-4 melalui <i>terminal</i> kemudian membuka tampilan <i>client interface</i> dari <i>controller</i> melalui <i>browser</i>
Masukan	<i>Path</i> ke file aplikasi <i>controller</i> , <i>port</i> pada tampilan <i>client interface</i>
Keluaran	Status pada <i>terminal</i> menunjukkan berhasil diinisialisasi dan tampilan <i>client interface</i> dapat dibuka
Hasil Uji Coba	BERHASIL

Proses ini bertujuan untuk menginisialisasi aplikasi *controller* agar dapat melakukan manajemen pada OpenFlow *switch*. Uji coba ini diawali dengan melakukan mengeksekusi aplikasi *controller* OpenDaylight. Berikut ini perintah yang diketikkan pada *terminal* linux ditunjukkan oleh gambar 5.3.

```
$ sudo ./run.sh
```

Gambar 5.3. Perintah untuk menjalankan *Controller*

Perintah tersebut akan langsung mengesekusi aplikasi *controller* dan memuat *plugin* yang ada didalamnya. Konfigurasi yang harus dilakukan sebelum mengeksekusi perintah ini dijelaskan pada BAB IV. Kemudian setelah inisialisasi *controller* selesai dilakukan, maka akan muncul tampilan seperti pada gambar 5.4 berikut ini.

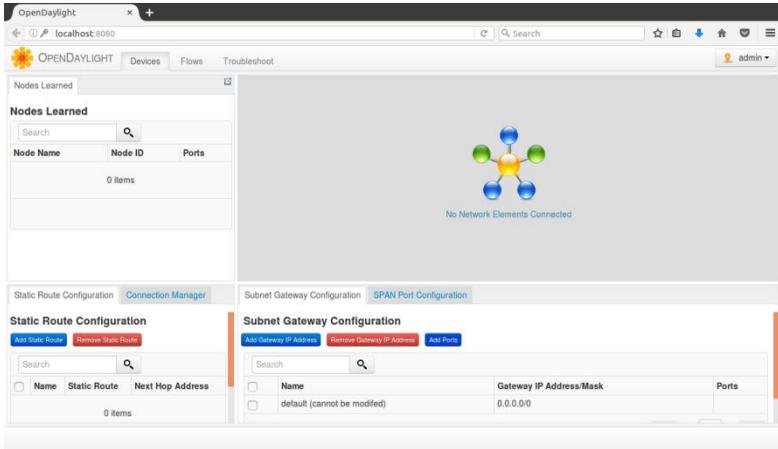
```

L.java:95) [bundlefile:na]
    at org.opendaylight.controller.sal.binding.impl.forward.DonForwardedBindingBrokerImpl.tryToDeployDonForwarder(DonForwardedBindingBrokerImpl.java:114) [bundlefile:na]
    at org.opendaylight.controller.sal.binding.impl.forward.DonForwardedBindingBrokerImpl$DonMountPointForwardingManager.onMountPointCreated(DonForwardedBindingBrokerImpl.java:137) [bundlefile:na]
    at org.opendaylight.controller.md.sal.don.broker.impl.mount.DonMountPointServiceImpl.notifyMountCreated(DonMountPointServiceImpl.java:48) [bundlefile:na]
    at org.opendaylight.controller.md.sal.don.broker.impl.mount.DonMountPointServiceImpl.registerMountPoint(DonMountPointServiceImpl.java:70) [bundlefile:na]
    at org.opendaylight.controller.md.sal.don.broker.impl.mount.DonMountPointServiceImpl$DonMountPointBuilderImpl.register(DonMountPointServiceImpl.java:110) [bundlefile:na]
    at org.opendaylight.controller.sal.connect.netconf.sal.NetConfDeviceSalProvider$MountInstance.onDeviceConnected(NetConfDeviceSalProvider.java:327) [bundlefile:na]
    at org.opendaylight.controller.sal.connect.netconf.sal.NetConfDeviceSalFacade.onDeviceConnected(NetConfDeviceSalFacade.java:93) [bundlefile:na]
    at org.opendaylight.controller.sal.connect.netconf.sal.NetConfDeviceSalFacade.onDeviceConnected(NetConfDeviceSalFacade.java:98) [bundlefile:na]
    at org.opendaylight.controller.sal.connect.netconf.NetConfDevice.handleSalInitializationSuccess(NetConfDevice.java:126) [bundlefile:na]
    at org.opendaylight.controller.sal.connect.netconf.NetConfDevice.access$800(NetConfDevice.java:54) [bundlefile:na]
    at org.opendaylight.controller.sal.connect.netconf.NetConfDevice$RecursiveSchemaSetup$1.onSuccess(NetConfDevice.java:325) [bundlefile:na]
    at org.opendaylight.controller.sal.connect.netconf.NetConfDevice$RecursiveSchemaSetup$1.onSuccess(NetConfDevice.java:320) [bundlefile:na]
    at com.google.common.util.concurrent.Futures$4.run(Futures.java:1149) [bundlefile:na]
    at com.google.common.util.concurrent.MoreExecutors$SameThreadExecutorService.execute(MoreExecutors.java:293) [bundlefile:na]
    at com.google.common.util.concurrent.ExecutionList$RunnableExecutorPair.execute(ExecutionList.java:190) [bundlefile:na]
    at com.google.common.util.concurrent.ExecutionList.add(ExecutionList.java:106) [bundlefile:na]
    at com.google.common.util.concurrent.AbstractFuture.addListener(AbstractFuture.java:170) [bundlefile:na]
    at com.google.common.util.concurrent.ForwardingListenerFuture.addListener(ForwardingListenerFuture.java:47) [bundlefile:na]
    at com.google.common.util.concurrent.Futures.addCallback(Futures.java:1152) [bundlefile:na]
    at com.google.common.util.concurrent.Futures.addCallback(Futures.java:1088) [bundlefile:na]
    at org.opendaylight.controller.sal.connect.netconf.NetConfDevice$RecursiveSchemaSetup.setupSchema(NetConfDevice.java:349) [bundlefile:na]
    at org.opendaylight.controller.sal.connect.netconf.NetConfDevice$RecursiveSchemaSetup.run(NetConfDevice.java:303) [bundlefile:na]
    at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511) [na:1.8.0_111]
    at java.util.concurrent.FutureTask.run(FutureTask.java:260) [na:1.8.0_111]
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142) [na:1.8.0_111]
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617) [na:1.8.0_111]
    at java.lang.Thread.run(Thread.java:745) [na:1.8.0_111]
2016-12-03 02:31:02.658 INFO [main-netconf-processor-processing-executor-3] INFO o.o.c.s.c.netconf.NetConfDevice - RemoteDevice(controller-config):
netconf connector initialized successfully
osgi>

```

Gambar 5.4. Tampilan *Log* Inisialisasi *Controller*

Controller OpenDaylight yang digunakan ini juga menyediakan tampilan antarmuka melalui *web* yang dapat diakses secara *default* melalui *port* 8080 dengan memasukkan *clientname* dan *password*. Gambar 5.5 berikut ini menunjukkan tampilan antarmuka dari *controller* *.opendaylight* melalui *browser*.



Gambar 5.5. Tampilan Antarmuka Controller

5.2.2. Inisialisasi *Load Switching*

Proses inisialisasi *load switching* merupakan proses menjalankan aplikasi *load switching* yang telah ditulis dalam bahasa python. Proses ini bertujuan untuk melakukan inisialisasi aplikasi *load switching* yang akan mengambil data statistik *throughput* dari *controller* melalui REST API. Keberhasilan proses ini ditandai dengan munculnya antarmuka aplikasi dalam *terminal* dan menunjukkan statistik *gateway*, *throughput* dari *controller port* saat ini. Tabel 3 berikut ini menunjukkan prosedur melakukan uji coba pada proses inisialisasi *load switching*.

Tabel 3. Uji Coba Inisialisasi *load switching*

ID	UJ-02
Nama	Uji Coba Proses Inisialisasi <i>load switching</i>
Tujuan Uji Coba	Menguji proses inisialisasi <i>load switching</i> yang mengambil data statistik <i>throughput controller</i>

	melalui REST API
Kondisi Awal	<i>load switching</i> belum berjalan, <i>Controller</i> sudah berjalan, OpenFlow <i>switch</i> sudah dihubungkan ke <i>controller</i>
Skenario	Menjalankan aplikasi <i>load switching</i> untuk mengambil statistik <i>throughput</i> ketika <i>controller</i> dan OpenFlow <i>switch</i> sudah saling terhubung.
Masukan	<i>Path</i> ke file aplikasi <i>load switching</i>
Keluaran	Muncul data statistik <i>throughput</i> dan status <i>gateway</i> pada <i>terminal</i>
Hasil Uji Coba	BERHASIL

Proses inisialisasi diawali dengan menjalankan perintah pada *terminal* linux. Selanjutnya aplikasi akan berjalan terus menerus selama tidak terdapat *interrupt* yang dilakukan oleh *administrator*. Gambar 5.6 berikut ini menunjukkan proses eksekusi perintah pada *terminal* linux.

```
$ python odlloadbalancing.py
```

Gambar 5.6. Perintah Inisialisasi *load switching*

Untuk menandai proses inisialisasi aplikasi *load switching* berhasil dilakukan maka dapat dilihat dari tampilan antarmuka *terminal* yang menunjukkan status *flow* yang sedang aktif, *gateway* dan statistik *throughput* saat ini. Data yang ditampilkan pada *terminal* ini akan terus berubah selama sistem tidak dihentikan atau terdapat *interrupt* yang mengharuskan aplikasi untuk berhenti.

```

pranawa@pranawa-HP-Pavillion-14-Notebook-PC: ~/TA/codingan
pranawa@pranawa-HP-Pavillion-14-Notebook-PC:~/TA/codingan$ python odltest.py
Enter master node (case sensitive with quotes ) : [nodeA/nodeB/nodeC/nodeD]:"nodeA"
master port : nodeA
Success

Node: nodeA
tx : 0.0 rx : 0.0

Node: nodeB
tx : 0.0 rx : 0.0

Node: nodeC
tx : 0.0 rx : 0.0

Node: nodeD
tx : 0.0 rx : 0.0

Node: nodeA
tx : 0.0 rx : 0.0

Node: nodeB
tx : 0.0 rx : 0.0

```

Gambar 5.7. Tampilan Antarmuka *load switching*

5.2.3. Pengambilan Statistik *Throughput*

Proses pengambilan statistik *throughput* merupakan sebuah proses otomatis yang akan berjalan ketika aplikasi *load switching* diinisialisasi. Proses ini bertujuan untuk mengambil statistik *throughput* pada *controller port* melalui REST API. Pengambilan statistik *throughput* ini akan menampilkan data *throughput* ketika OpenFlow *switch* sudah dilakukan konfigurasi dan terhubung pada *controller*. Tabel 4 berikut ini menunjukkan prosedur uji coba proses pengambilan statistik *throughput* pada *controller*.

Tabel 4. Uji Coba Pengambilan Statistik *Throughput*

ID	UJ-03
Nama	Uji Coba Proses Pengambilan Statistik <i>Throughput</i>
Tujuan Uji	Menguji proses pengambilan statistik

Coba	<i>throughput</i> melalui REST API ketika aplikasi <i>load switching</i> sudah selesai di inisialisasi
Kondisi Awal	Pengambilan <i>throughput</i> belum berjalan, OpenFlow <i>switch</i> sudah dihubungkan ke <i>controller</i>
Skenario	Menjalankan aplikasi <i>load switching</i> untuk mengambil statistik <i>throughput</i> ketika <i>controller</i> dan OpenFlow <i>switch</i> sudah saling terhubung.
Masukan	<i>Path</i> ke aplikasi <i>load switching</i>
Keluaran	Muncul perubahan data statistik <i>throughput</i> pada masing-masing <i>port</i>
Hasil Uji Coba	BERHASIL

Tampilan antarmuka proses pengambilan statistik *throughput* ini dapat dilihat pada *terminal* linux yang menampilkan data *throughput transfer bytes*, dan *received bytes* setiap detik seperti ditunjukkan pada gambar 5.8

```

pranawa@pranawa-HP-Pavilion-14-Notebook-PC: ~/TA/codingan
Node: nodeB
tx : 0.0 rx : 0.0

Node: nodeC
tx : 0.0 rx : 0.0

Node: nodeD
tx : 0.0 rx : 0.0

Node: nodeA
tx : 0.158203125 rx : 0.271484375

Node: nodeB
tx : 0.158203125 rx : 0.271484375

Node: nodeC
tx : 0.158203125 rx : 0.271484375

Node: nodeD
tx : 0.158203125 rx : 0.271484375

No modification detected
==> gateway : node A

```

Gambar 5.8. Tampilan Antarmuka Proses Pengambilan Statistik *Throughput*

5.2.4. Memasukkan Data ke *Database*

Sama seperti proses pengambilan *throughput*, proses memasukkan data kedalam *database* juga merupakan sebuah proses yang otomatis dilakukan ketika aplikasi *load switching* selesai diinisialisasi. Proses ini bertujuan untuk menyimpan data statistik *throughput* yang terdapat pada tabel *throughput controller* yang kemudian akan digunakan sebagai *parameter* penentuan *flow*. Proses ini akan memanfaatkan API yang disediakan oleh *database* influxdb yang dapat dieksekusi melalui kode program. Kondisi awal pada proses uji coba memasukkan data ke *database* ini adalah kondisi tabel pada *database* yang masih kosong/belum terdapat data hasil pengambilan statistik terbaru. Tabel 5 berikut ini menunjukkan prosedur melakukan uji coba memasukkan data statistik *load switching*.

Tabel 5. Uji Coba Memasukkan Data *Throughput* ke *Database*

ID	UJ-04
Nama	Uji Coba Proses <i>Insert Data Throughput</i> kedalam <i>Database</i>
Tujuan Uji Coba	Menguji proses <i>insert data throughput</i> kedalam <i>database</i>
Kondisi Awal	<i>Database</i> masih kosong
Skenario	Menjalankan aplikasi <i>load switching</i> yang akan mengambil data statistik <i>throughput</i> setiap detik kemudian memasukkan kedalam <i>database</i> influxdb melalui API
Masukan	<i>Path</i> ke aplikasi <i>load switching</i>
Keluaran	<i>Database</i> influxdb berisi data <i>throughput</i>
Hasil Uji Coba	BERHASIL

Hasil keluaran dari proses memasukkan data kedalam *database* ini adalah adanya data baru yang tersimpan pada suatu tabel di dalam *database*. Tabel ini akan berisi waktu saat data dimasukkan, *host router*, *transfer bytes*, dan *received bytes* pada *host router* yang terhubung pada *port* tersebut. Gambar 5.9 berikut ini menunjukkan antarmuka proses memasukkan data *throughput* kedalam *database* setelah berhasil dilakukan.

```

pranawa@pranawa-HP-Pavilion-14-Notebook-PC: ~/TA/codingan
time          host          rx            tx
----          -
1482382420800938886  nodeA        0.154296875  0.05859375
1482382420861755416  nodeB        0.29296875   0.275390625
1482382420906400379  nodeC        0.29296875   0.275390625
1482382420950533515  nodeD        0.154296875  0.05859375
1482382425729080815  nodeA        0.1171875    0.08203125
1482382425798472126  nodeB        0.29296875   0.29296875
1482382425842544002  nodeC        0.29296875   0.29296875
1482382425887094270  nodeD        0.05859375   0.041015625
1482382430748276638  nodeA        0.212890625  0.1171875
1482382430826362047  nodeB        0.388671875  0.275390625
1482382430883511160  nodeC        0.234375     0.234375
1482382430936536405  nodeD        0.212890625  0.1171875
1482382435851821606  nodeB        0.29296875   0.29296875
1482382435896679629  nodeC        0.29296875   0.29296875
1482382435949525419  nodeD        0.05859375   0.041015625
1482382440757697519  nodeA        0.1171875    0.1171875
1482382440822668772  nodeB        0.388671875  0.29296875
1482382440866684835  nodeC        0.3515625    0.333984375
1482382440911410634  nodeD        0.212890625  0.1171875
1482382445704825639  nodeB        0.234375     0.234375
1482382445761791864  nodeC        0.234375     0.234375
1482382445816221237  nodeD        0.05859375   0.041015625

```

Gambar 5.9. Antarmuka Proses Memasukkan Data *Throughput* ke *Database*

5.2.5. Penentuan *Flow*

Proses penentuan *flow* adalah sebuah proses yang dilakukan oleh *controller* untuk menentukan aliran data dari sebuah paket. Proses penentuan *flow* ini ditentukan berdasarkan parameter *throughput* terbaik dengan menggunakan data *throughput* yang telah diambil pada proses pengambilan data statistik *throughput* dan memasukkannya kedalam *database*. Proses ini juga akan melakukan pemasangan *flow* pada perangkat OpenFlow *switch* yang terhubung dengan *controller* dan memiliki

node ID tertentu. Tabel 6 dibawah ini menunjukkan prosedur uji coba penentuan *flow* berdasarkan parameter *throughput* terbaik.

Tabel 6. Uji Coba Proses Penentuan *Flow*

ID	UJ-05
Nama	Uji Coba Proses Penentuan <i>Flow</i>
Tujuan Uji Coba	Menguji proses penentuan <i>flow</i> dan memasangkannya kedalam OpenFlow <i>switch</i> berdasarkan <i>throughput</i> terbaik
Kondisi Awal	<i>Flow table</i> pada OpenFlow <i>switch</i> masih kosong
Skenario	Menjalankan aplikasi <i>load switching</i> yang akan menentukan parameter <i>throughput</i> terbaik dan data <i>flow</i> kemudian mengamati perubahan <i>flow</i> apabila terjadi perubahan <i>throughput</i>
Masukan	Path ke file aplikasi <i>load switching</i>
Keluaran	Status gateway pada <i>terminal</i> , bertambahnya <i>flow</i> pada <i>flow table</i> OpenFlow <i>switch</i>
Hasil Uji Coba	BERHASIL

Gambar 5.10 sampai dengan 5.12 berikut ini menjelaskan tentang proses uji fungsionalitas penentuan *flow*. Gambar 5.10 merupakan *master node* yang akan menjadi *flow* awal dan harus ditentukan terlebih dahulu sebelum sistem *load switching* dijalankan. Gambar 5.11 merupakan hasil perubahan *flow* dengan memilih *host router* yang memiliki nilai parameter terbaik.

```

pranawa@pranawa-HP-Pavilion-14-Notebook-PC: ~/TA/codingan
pranawa@pranawa-HP-Pavilion-14-Notebook-PC:~/TA/codingan$ python odltest.py
Enter master node (case sensitive with quotes ) : [nodeA/nodeB/nodeC/nodeD]:"nodeA"
master port : nodeA
Success

Node: nodeA
tx : 0.0 rx : 0.0

Node: nodeB
tx : 0.0 rx : 0.0

Node: nodeC
tx : 0.0 rx : 0.0

Node: nodeD
tx : 0.0 rx : 0.0

Node: nodeA
tx : 0.0 rx : 0.0

Node: nodeB
tx : 0.0 rx : 0.0

```

Gambar 5.10. Tampilan Penentuan Master Port *Flow* pada *Terminal Linux*

```

pranawa@pranawa-HP-Pavilion-14-Notebook-PC: ~/TA/codingan

Node: nodeC
tx : 0.0 rx : 0.0

Node: nodeD
tx : 0.0 rx : 0.0

Node: nodeA
tx : 0.236328125 rx : 0.154296875

Node: nodeB
tx : 0.140625 rx : 0.05859375

Node: nodeC
tx : 0.140625 rx : 0.05859375

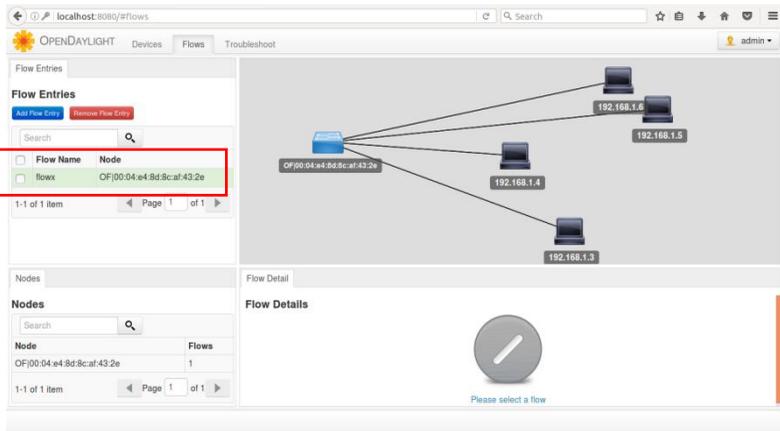
Node: nodeD
tx : 0.140625 rx : 0.4287109375

Success
==> gateway : node D

Node: nodeA
tx : 0.0 rx : 0.0

```

Gambar 5.11. Tampilan Perpindahan *Flow* pada *Terminal Linux*



Gambar 5.12. Tampilan Penambahan *Flow* pada *Browser*

5.2.6. Proses *Dial-Up*

Proses *dial-up* merupakan sebuah proses menghubungkan sebuah perangkat dengan *internet* melalui jaringan telepon seluler. Proses *dial-up* ini bertujuan untuk memberikan akses menuju ke *internet* dari *host router* yang digunakan sebagai *gateway*. Setelah *host router* selesai melakukan proses *dial-up*, maka dilakukan uji coba dengan mengirimkan paket ICMP melalui PING kepada sebuah alamat IP di *internet*. Tabel 7 berikut ini menunjukkan prosedur uji coba proses *dial-up* dari sebuah *modem* pada *host router*.

Tabel 7. Uji Coba Proses *Dial-Up*

ID	UJ-06
Nama	Uji Coba Proses <i>Dial-Up</i>
Tujuan Uji Coba	Menguji proses <i>dial-up</i> untuk menyambungkan koeneksi <i>host router</i> dengan <i>internet</i> melalui Status pada terminal menunjukkan berhasil

	diinisialisasi dan tampilan <i>client</i> interface dapat dibukamodem GSM
Kondisi Awal	<i>Host router</i> belum tersambung ke <i>internet</i>
Skenario	Menjalankan aplikasi <i>wvdial</i> pada <i>host router</i> kemudian mengirimkan paket ICMP kepada alamat 8.8.8.8
Masukan	Sintaks program <i>wvdial</i>
Keluaran	Status koneksi pada <i>terminal</i> , paket ICMP mendapatkan <i>reply</i>
Hasil Uji Coba	BERHASIL

Parameter keberhasilan dari proses *dial-up* ini adalah keluarnya jendela *terminal* yang menunjukkan *log* proses *dial-up* dan status koneksi. Apabila *host router* berhasil melakukan proses *dial-up* maka tampilan antarmuka pada jendela *terminal* akan menunjukkan bahwa *host router* berhasil mendapatkan alamat IP dan alamat DNS (*Domain Name System*).

```

pi@raspberrypi: ~
--> Modem initialized.
--> Sending: ATDT*99#
--> Waiting for carrier.
ATDT*99#
CONNECT 10000000
--> Carrier detected. Starting PPP immediately.
--> Starting pppd at Thu Dec 22 06:47:46 2016
--> Pid of pppd: 1574
--> Using interface ppp0
--> pppd: *(w[01]0w[01]
--> local IP address 172.24.37.1
--> pppd: *(w[01]0w[01]
--> remote IP address 10.64.64.64
--> pppd: *(w[01]0w[01]
--> primary DNS address 114.5.5.77
--> pppd: *(w[01]0w[01]
--> secondary DNS address 114.5.5.5
--> pppd: *(w[01]0w[01]

```

Gambar 5.13. Tampilan Antarmuka Proses *Dial-Up*

5.2.7. Forwarding Paket

Proses *forwarding* paket merupakan sebuah proses pemberian beberapa aturan (*policy*) pada *host router* kepada paket yang masuk. Konfigurasi proses *forwarding* ini dilakukan ketika *host router* pertamakali dijalankan secara otomatis. Untuk menguji apakah proses *forwarding* paket ini dapat berjalan dengan baik, maka dilakukan pengujian dengan mengirimkan paket ICMP menggunakan PING dari *client* ke *internet* melalui *host router*. Tabel 8 berikut ini menunjukkan prosedur uji coba proses *forwarding* paket yang dilakukan oleh *host router*.

Tabel 8. Uji Coba Proses Forwarding Paket

ID	UJ-07
Nama	Uji Coba Proses <i>Forwarding</i> Paket
Tujuan Uji Coba	Menguji proses <i>forwarding</i> paket dari <i>client</i> ke <i>internet</i> melalui <i>host router</i>
Kondisi Awal	<i>Client</i> belum terhubung ke <i>internet</i> , <i>host router</i> sudah terhubung ke <i>internet</i>
Skenario	Menuliskan sintaks <i>forwarding</i> pada <i>host router</i> kemudian melakukan pengaturan <i>gateway</i> pada <i>client</i> dan mengirimkan paket ICMP kepada alamat 8.8.8.8
Masukan	Sintaks PING 8.8.8.8
Keluaran	Paket ICMP mendapatkan <i>reply</i>
Hasil Uji Coba	BERHASIL

Parameter keberhasilan uji coba ini adalah *client* yang mengirimkan paket ICMP mendapatkan *reply* dari alamat IP yang dituju. Gambar 5.14 berikut ini menunjukkan *screenshot* uji coba *client* yang berhasil mendapatkan *reply* dari sebuah alamat IP.

```

pranawa@pranawa-MS-7759: ~
pranawa@pranawa-MS-7759:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data:
64 bytes from 8.8.8.8: icmp_seq=1 ttl=47 time=66.4 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=47 time=64.5 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=47 time=83.0 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=47 time=86.0 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=47 time=62.9 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=47 time=69.9 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=47 time=65.2 ms
^C
--- 8.8.8.8 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6007ms
rtt min/avg/max/mdev = 62.981/71.199/86.013/8.714 ms
pranawa@pranawa-MS-7759:~$

```

Gambar 5.14. Tampilan Antarmuka Uji Coba *Forwarding* Paket

5.2.8. Pengambilan dan Pengiriman *Throughput*

Proses pengambilan dan pengiriman *throughput* merupakan sebuah proses untuk mengambil data *throughput* dari koneksi jaringan seluler yang sedang digunakan. Proses ini bertujuan untuk mengambil dan mengirim data *throughput* kemudian mengirimkan data tersebut kepada *controller* yang akan memodifikasi tabel *throughput* pada *controller*. Tabel 9 berikut ini menunjukkan prosedur uji coba proses pengambilan dan pengiriman *throughput*.

Tabel 9. Uji Coba Pengambilan dan Pengiriman *Throughput*

ID	UJ-08
Nama	Uji Coba Proses Pengambilan dan Pengiriman <i>Throughput</i>
Tujuan Uji	Menguji proses pengambilan dan pengiriman <i>throughput</i> untuk memodifikasi tabel

Coba	<i>throughput</i> pada <i>controller</i> .
Kondisi Awal	Aplikasi pengambilan dan pengiriman <i>throughput</i> belum dijalankan, tabel <i>throughput</i> pada <i>controller</i> belum terdapat perubahan
Skenario	Menjalankan aplikasi pengambilan dan pengiriman <i>throughput</i>
Masukan	<i>Path</i> ke aplikasi <code>gettp.py</code>
Keluaran	Status <i>throughput</i> pada terminal, terdapat perubahan pada tabel <i>throughput</i> di <i>controller</i>
Hasil Uji Coba	BERHASIL

Proses pengambilan dan pengiriman *throughput* ini akan menampilkan status koneksi pada antarmuka pada *terminal*. Status koneksi yang ditampilkan adalah status koneksi antara alamat IP tujuan yang statis dan jumlah *bytes* yang dikirim. Jumlah *bytes* yang dikirim ini dihitung berdasarkan rumus pada BAB III. Gambar 5.15 berikut ini menunjukkan proses pengambilan dan pengiriman data *throughput* yang berhasil dilakukan.

Pada saat implementasi, tampilan antarmuka tidak akan terlihat karena proses pengambilan dan pengiriman data *throughput* ini akan berjalan di *background process*. Akan tetapi perubahan data *throughput* tetap dapat dilihat pada tabel statistik *port* pada *controller*.

```

pi@raspberrypi: ~
pi@raspberrypi:~ $ python gettcp.py
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=48 time=285 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 285.776/285.776/285.776/0.000 ms
PING 192.168.1.4 (192.168.1.4) 0(28) bytes of data.
8 bytes from 192.168.1.4: icmp_seq=1 ttl=64

--- 192.168.1.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
0.893617021277

PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=48 time=326 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 326.875/326.875/326.875/0.000 ms
PING 192.168.1.4 (192.168.1.4) 0(28) bytes of data.
8 bytes from 192.168.1.4: icmp_seq=1 ttl=64

```

Gambar 5.15. Tampilan Antarmuka Proses Pengambilan dan Pengiriman *Throughput* pada *Terminal Linux*

The screenshot shows the OpenDaylight Troubleshoot web interface. The main content area displays a node with ID OF:00:04:e4:8d:8c:af:43:2e. Below this, there is a table titled 'Ports' which is highlighted with a red box. This table provides detailed throughput statistics for various network connectors.

Node Connector	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs	Rx CRC Errs	Collisions
OF1@	88	15	12793	740	0	0	-1	-1	-1	-1	-1	-1
OF2@	79	14	11922	680	0	0	-1	-1	-1	-1	-1	-1
OF3@	77	13	11765	620	0	0	-1	-1	-1	-1	-1	-1
OF4@	77	15	11748	740	0	0	-1	-1	-1	-1	-1	-1
OF5@	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1

The screenshot shows the OpenDaylight Troubleshoot interface. The main area displays a node with ID OF:00:04:e4:8d:8c:af:43:2e. Below this, a table titled 'Ports' is highlighted with a red border. The table contains the following data:

Node Connector	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs	Rx CRC Errs	Collisions
OF1@	97	29	14010	1456	0	0	-1	-1	-1	-1	-1	-1
OF2@	97	29	13603	1438	0	0	-1	-1	-1	-1	-1	-1
OF3@	94	28	13348	1378	0	0	-1	-1	-1	-1	-1	-1
OF4@	94	29	13369	1456	0	0	-1	-1	-1	-1	-1	-1
OF5@	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1

Gambar 5.16. Tampilan Antarmuka Perubahan Rx Bytes pada Browser

5.2.9. Inisialisasi *Network Monitoring*

Proses inisialisasi *network monitoring* merupakan sebuah proses untuk menjalankan antarmuka *network monitoring*. Aplikasi yang digunakan untuk melakukan monitor ini menggunakan aplikasi grafana dengan menggunakan database influxdb. Tujuan dari penggunaan antarmuka ini adalah untuk memudahkan dalam pengamatan jaringan, karena antarmuka ini akan melakukan representasi data *throughput* menjadi diagram secara otomatis dan *realtime*. Tabel 10 berikut ini menunjukkan prosedur uji coba proses inisialisasi *network monitoring*.

Tabel 10. Uji Coba Inisialisasi *Network Monitoring*

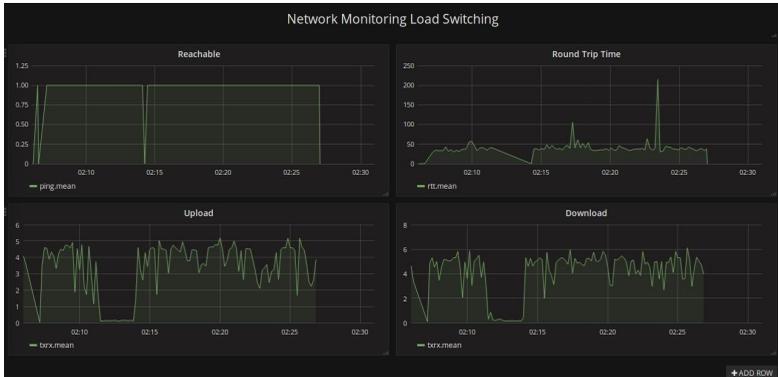
ID	UJ-09
Nama	Uji Coba Proses Inisialisasi <i>Network Monitoring</i>
Tujuan Uji Coba	Menguji proses inisialisasi antarmuka <i>network monitoring</i> menggunakan grafana

Kondisi Awal	Antarmuka <i>network monitoring</i> belum dijalankan, sudah terdapat <i>database throughput</i>
Skenario	Menjalankan <i>server grafana</i> kemudian mengakses antarmuka melalui <i>browser</i> dengan <i>port 3000</i>
Masukan	Sintaks menjalankan <i>grafana-server</i>
Keluaran	Antarmuka <i>network monitoring</i> dapat diakses melalui <i>browser</i>
Hasil Uji Coba	BERHASIL

Ketika proses inialisasi *network monitoring* pertama kali dijalankan, maka langkah awal yang harus dilakukan adalah menjalankan *server* aplikasi. Gambar 5.17 berikut ini menunjukkan kode yang dieksekusi pada *terminal* ketika antarmuka *network monitoring* pertama kali dijalankan.

```
$ sudo service grafana-server start
```

Gambar 5.17. Perintah Melakukan Inialisasi *Network Monitoring*
Setelah *server network monitoring* selesai dijalankan, maka antarmuka dapat diakses melalui *browser* dengan menggunakan *port 3000*. Sebelum dapat melihat representasi data menggunakan diagram, maka perlu dilakukan konfigurasi dari sistem antarmuka *network monitoring* seperti yang dijelaskan pada BAB IV. Gambar 5.18 berikut ini menunjukkan tampilan antarmuka *network moitoring* yang diakses melalui *browser*.



Gambar 5.18. Tampilan Antarmuka *Network Monitoring*

5.3. Uji Coba Performa

Pengujian performa pada sistem ini dilakukan untuk mengetahui seberapa baik kualitas *throughput* yang didapat oleh pengguna sebelum dan sesudah menggunakan mekanisme *load switching* menggunakan *software defined network* dengan *controller* OpenDaylight. Uji coba performa yang dilakukan meliputi pengukuran *network reachable*, pengukuran *round trip time*, pengukuran pengunduhan berkas, dan pengukuran unggah berkas.

Pengukuran *network reachable* digunakan untuk mengetahui tingkat ketersampaian sebuah paket kepada sebuah alamat komputer. Pengukuran *round trip time* digunakan untuk mengetahui waktu dari sebuah paket dapat dikembalikan oleh komputer tujuan. Pengukuran pengunduhan berkas dilakukan untuk mengukur kemampuan jaringan dalam melakukan pengunduhan berkas dengan ukuran tertentu. Pengukuran pengunggahan berkas dilakukan untuk mengukur kemampuan jaringan dalam melakukan pengunggahan berkas dengan ukuran tertentu.

Uji coba performa tersebut diterapkan pada kondisi jaringan yang berpindah dari satu lokasi ke lokasi yang lain secara

terus menerus. Uji coba performa juga diterapkan dengan menggunakan sebuah alamat komputer dengan alamat yang tetap, sehingga didapatkan sebuah data dengan variabel yang sama. Alamat IP yang digunakan uji coba adalah alamat `freeshare.lp.if.its.ac.id`.

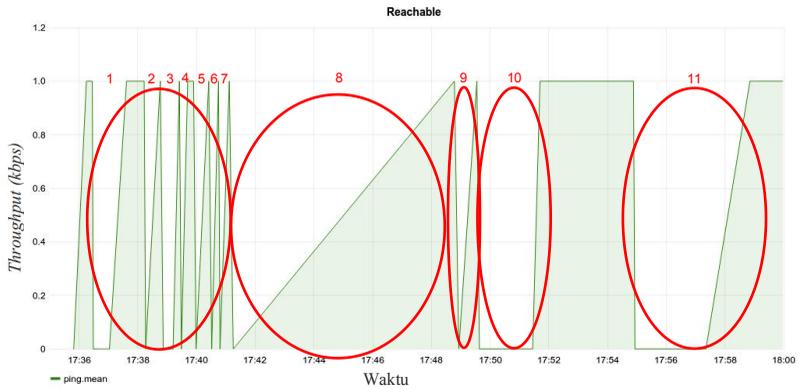
5.3.1. Uji Performa *Network Reachable*

Uji performa *network reachable* merupakan uji performa yang dilakukan untuk mengetahui apakah sebuah jaringan mampu mengirim dan menerima data. Pada uji performa ini dilakukan dengan 2 (dua) tahap, yaitu uji performa *network reachable* tanpa *load switching* dan uji performa *network reachable* dengan *load switching*.

5.3.1.1. Uji Performa *Network Reachable* tanpa *Load Switching*

Uji performa *network reachable* tanpa *load switching* dilakukan dengan cara menguji koneksi antara 2 (dua) *host* menggunakan jaringan *internet*. Jaringan yang dipakai berdasarkan sinyal terbaik pada rute yang telah ditentukan sebelumnya.

Pada uji performa *network reachable* ini dilakukan untuk mengetahui tingkat *reliability* sebuah jaringan dengan koneksi 1 (satu) *provider* yaitu Telkomsel. Gambar 5.19 berikut ini menunjukkan grafik hasil uji performa *network reachable*.



Gambar 5.19. Grafik Hasil Uji Performa *Network Reachable* tanpa *Load Switching*

Berdasarkan grafik hasil uji performa *network reachable* tanpa *load switching* diatas, dapat dilihat bahwa terdapat 11 (sebelas) titik *unreachable* yang ditunjukkan pada nomor yang berwarna merah. *Delay* yang dihasilkan dari uji performa *network reachable* tanpa *load switching* ditunjukkan pada tabel 11 dibawah ini.

Tabel 11. Tabel *Delay Time* pada Uji Coba *Network Reachable* tanpa *Load Switching*

Nomor	<i>Delay Time (second)</i>
1	70
2	34
3	38
4	18
5	30
6	20
7	22
8	460
9	46
10	130

11	238
total	1106

Berdasarkan tabel 11 diatas dapat diketahui bahwa jumlah keseluruhan *delay time* pada uji performa tanpa *load switching* adalah 1106 detik.

5.3.1.2. Uji Performa *Network Reachable* dengan *Load Switching* tanpa Waktu Tunggu

Uji performa *network reachable* dengan *load switching* tanpa waktu tunggu dilakukan dengan cara menguji koneksi antara 2 (dua) *host* menggunakan jaringan internet pada 4 operator. Keempat operator ini akan dihitung parameter pemilihan jalur terbaiknya, kemudian paket yang berasal dari *client* akan diteruskan melalui operator yang telah dipilih tersebut. Pemilihan jalur terbaik dilakukan pada setiap iterasi pengecekan *throughput*.

Pada uji performa *network reachable* ini dilakukan untuk mengetahui tingkat *reliability* sebuah jaringan dengan koneksi 4 (empat) *provider*. Uji performa ini menggunakan *load switching* dengan pengecekan parameter *throughput* dan perpindahan *flow* setiap iterasi. Gambar 5.20 berikut ini menunjukkan grafik hasil uji performa *network reachable*.



Gambar 5.20. Grafik Hasil Uji Performa *Network Reachable* dengan *Load Switching* tanpa Waktu Tunggu

Berdasarkan grafik hasil uji performa *network reachable* dengan *load switching* tanpa waktu tunggu dapat dilihat bahwa terdapat 1 (satu) titik *network unreach* yang ditunjukkan pada nomor yang berwarna merah. *Delay time* yang dihasilkan dari uji performa *network reachable* dengan *load switching* tanpa waktu tunggu ini ditunjukkan pada tabel 12 berikut ini.

Tabel 12. Tabel *Delay Time* pada Uji Performa *Network Reachable* dengan *Load Switching* tanpa Waktu Tunggu

Nomor	<i>Delay Time (second)</i>
1	170
total	170

Berdasarkan data *delay time* diatas dapat diketahui bahwa jumlah *delay time* yang dihasilkan dari uji performa *network reachable* dengan *load switching* tanpa waktu tunggu adalah 170 detik.

5.3.1.3. Uji Performa *Network Reachable* dengan *Load Switching* menggunakan Waktu Tunggu

Uji performa *network reachable* dengan *load switching* menggunakan waktu tunggu dilakukan dengan cara menguji koneksi antara 2 (dua) *host* menggunakan jaringan internet pada 4 operator. Keempat operator ini akan dihitung parameter pemilihan jalur terbaiknya, kemudian paket yang berasal dari *client* akan diteruskan melalui operator yang telah dipilih tersebut. Pemilihan parameter dilakukan pada setiap 5 (lima) iterasi pengecekan *throughput*.

Pada uji performa *network reachable* ini dilakukan untuk mengetahui tingkat *reliability* sebuah jaringan dengan koneksi 4 (empat) *provider*. Uji performa ini menggunakan *load switching* dengan pengecekan parameter *throughput* dan perpindahan *flow* setiap 5 (lima) iterasi. Gambar 5.21 berikut ini menunjukkan grafik hasil uji performa *network reachable*.



Gambar 5.21. Grafik Hasil Uji Performa *Network Reachable* dengan *Load Switching* menggunakan Waktu Tunggu

Berdasarkan grafik hasil uji performa *network reachable* dengan *load switching* tanpa waktu tunggu dapat dilihat bahwa terdapat 2 (dua) titik *network unreach* yang ditunjukkan pada nomor yang bewarna merah.

Hasil *delay time* yang diperoleh dari uji performa *network reachable* dengan *load switching* menggunakan waktu tunggu ditunjukkan pada tabel 13 berikut ini.

Tabel 13. Tabel *Delay Time* pada Uji Performa *Network Reachable* dengan *Load Switching* menggunakan Waktu Tunggu

Nomor	<i>Delay Time (second)</i>
1	40
2	20
total	60

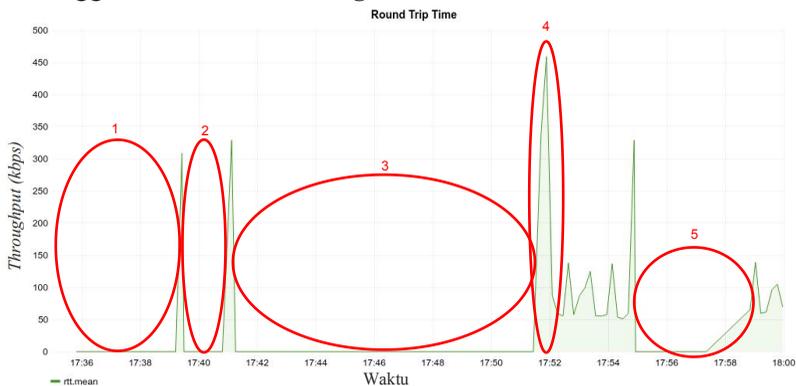
Berdasarkan data pada tabel *delay time* diatas dapat diketahui bahwa uji performa *network reachable* dengan *load switching* menggunakan waktu tunggu menghasilkan jumlah *delay time* 60 detik.

5.3.2. Uji Performa *Round Trip Time*

Uji performa *round trip time* (RTT) dilakukan untuk mengetahui berapa waktu yang dibutuhkan untuk mengirimkan paket tersebut. *Round trip time* dapat memiliki nilai yang bervariasi dikarenakan rute yang dilalui oleh sebuah paket dapat berubah setiap saat. *Round trip time* juga sangat tergantung dengan *reliability* dari suatu jaringan. Uji performa *round trip time* dibagi menjadi 2 (dua) tahap, yaitu uji performa *round trip time* tanpa *load switching* dan uji performa *round trip time* dengan *load switching*.

5.3.2.1. Uji Performa *Round Trip Time* tanpa *Load Switching*

Pada skenario uji performa *round trip time* tanpa *load switching*, suatu *client* akan mencoba mengirimkan protokol ICMP kepada sebuah alamat IP. Berdasarkan paket tersebut akan diperoleh status *log* mengenai waktu yang dibutuhkan agar paket tersebut sampai pada pengirim. Gambar 5.22 berikut ini menunjukkan grafik hasil uji performa *round trip time* tanpa menggunakan *load switching*.



Gambar 5.22. Grafik Hasil Uji Performa *Round Trip Time* tanpa menggunakan *Load Switching*

Berdasarkan data pada grafik hasil uji performa *round trip time* tanpa menggunakan *load switching* diperoleh 5 (lima) titik jaringan tidak *reliable* yang ditunjukkan pada poin yang berwarna merah. Pada poin 1, 2, 3 dan 5 menunjukkan suatu jaringan tidak dapat menjangkau komputer yang menjadi tujuan suatu paket. Poin 4 menunjukkan suatu jaringan lambat dalam mentransmisikan data karena memiliki *round trip* yang lebih tinggi daripada detik lainnya yaitu lebih dari 400 *milisecond*.

Waktu *delay time* yang dihasilkan dari uji performa *round trip time* tanpa menggunakan *load switching* dijelaskan pada tabel 14 berikut ini.

Tabel 14. Tabel *Delay Time* pada Uji Coba *Round Trip Time* tanpa *Load Switching*

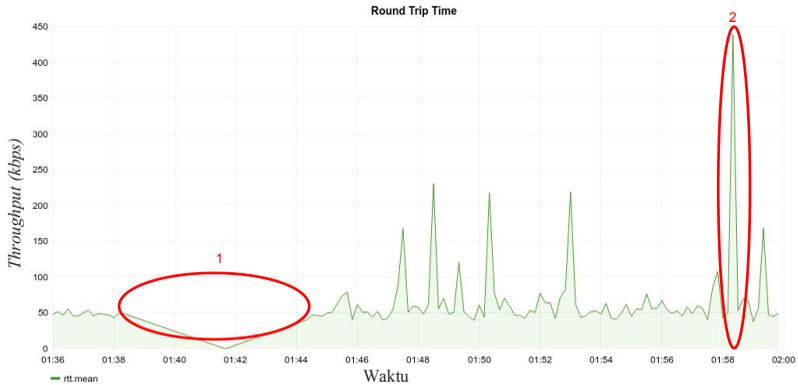
Nomor	<i>Delay Time (second)</i>
1	264
2	102
3	636
4	12
5	218
total	1232

Berdasarkan tabel 12 diatas dapat diketahui bahwa waktu *delay time* pada uji performa *round trip time* tanpa *load switching* adalah 1232 detik.

5.3.2.2. Uji Performa *Round Trip Time* dengan *Load Switching* tanpa Waktu Tunggu

Pada skenario uji performa *round trip time* dengan *load switching* tanpa waktu tunggu, suatu *client* akan mencoba mengirimkan protokol ICMP kepada sebuah alamat IP. Berdasarkan paket tersebut akan diperoleh status *log* apakah sebuah paket mampu dijawab oleh *server* beserta waktu yang dibutuhkan agar paket tersebut sampai pada pengirim. Gambar

5.23 berikut ini menunjukkan grafik hasil uji performa *round trip time* menggunakan *load switching* pada 4 (empat) *provider* dengan pemilihan parameter dan pengecekan *throughput* setiap iterasi.



Gambar 5.23. Grafik Hasil Uji Performa *Round Trip Time* dengan *Load Switching* tanpa Waktu Tunggu

Berdasarkan data pada grafik hasil uji performa *round trip time* menggunakan *load switching* tanpa waktu tunggu diperoleh 2 (dua) titik jaringan tidak *reliable* yang ditunjukkan pada poin yang berwarna merah. Pada poin 1 menunjukkan suatu jaringan tidak dapat menjangkau komputer yang menjadi tujuan suatu paket. Poin 2 menunjukkan suatu jaringan lambat dalam mentransmisikan data karena memiliki *round trip* yang lebih tinggi daripada detik lainnya yaitu lebih dari 400 *milisecond*.

Hasil *delay time* yang diperoleh dari uji performa *round trip time* dengan *load switching* tanpa waktu tunggu ditunjukkan pada tabel 15 berikut ini.

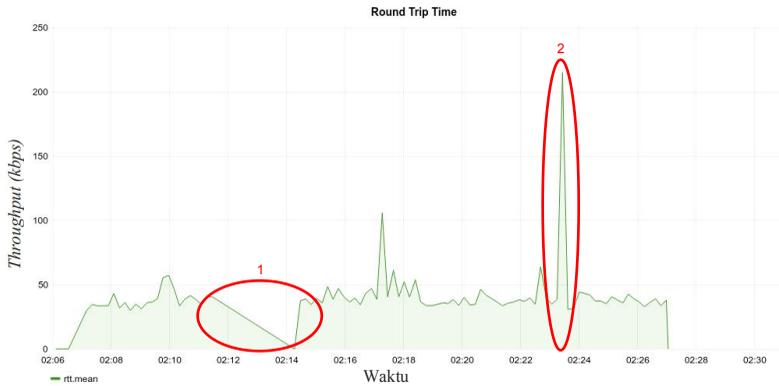
Tabel 15. Tabel *Delay Time* pada Uji Performa *Round Trip Time* dengan *Load Switching* tanpa Waktu Tunggu

Nomor	<i>Delay Time (second)</i>
1	360
2	20
total	380

Berdasarkan data pada tabel *delay time* diatas dapat diketahui bahwa uji performa *round trip time* dengan *load switching* tanpa waktu tunggu menghasilkan jumlah *delay time* 380 detik.

5.3.2.3. Uji Performa *Round Trip Time* dengan *Load Switching* menggunakan Waktu Tunggu

Pada skenario uji performa *round trip time* dengan *load switching* menggunakan waktu tunggu, suatu *client* akan mencoba mengirimkan protokol ICMP kepada sebuah alamat IP. Berdasarkan paket tersebut akan diperoleh status *log* apakah sebuah paket mampu dijawab oleh *server* beserta waktu yang dibutuhkan agar paket tersebut sampai pada pengirim. Gambar 5.24 berikut ini menunjukkan grafik hasil uji performa *round trip time* menggunakan *load switching* pada 4 (empat) *provider* dengan pemilihan parameter dan pengecekan *throughput* setiap 5 (lima) iterasi.



Gambar 5.24. Grafik Hasil Uji Performa *Round Trip Time* dengan *load switching* menggunakan Waktu Tunggu

Berdasarkan data pada grafik hasil uji performa *round trip time* menggunakan *load switching* dengan waktu tunggu diperoleh 2 (dua) titik jaringan tidak *reliable* yang ditunjukkan pada poin yang berwarna merah. Pada poin 1 menunjukkan suatu jaringan tidak dapat menjangkau komputer yang menjadi tujuan suatu paket. Poin 2 menunjukkan suatu jaringan lambat dalam mentransmisikan data karena memiliki *round trip* yang lebih tinggi daripada detik lainnya yaitu lebih dari 200 *milisecond*.

Hasil *delay time* yang diperoleh dari uji performa *round trip time* menggunakan *load switching* dengan waktu tunggu ditunjukkan pada tabel 16 berikut ini.

Tabel 16. Tabel *Delay Time* pada Uji Performa *Round Trip Time* menggunakan *Load Switching* dengan Waktu Tunggu

Nomor	<i>Delay Time (second)</i>
1	180
2	20
total	200

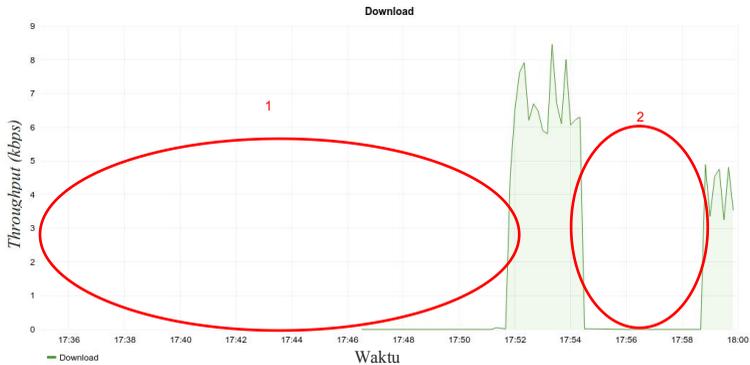
Berdasarkan data pada tabel *delay time* diatas dapat diketahui bahwa uji performa *round trip time* menggunakan *load switching* dengan waktu tunggu menghasilkan jumlah *delay time* 200 detik.

5.3.3. Uji Performa Pengunduhan

Uji performa pengunduhan merupakan proses yang dilakukan untuk mengetahui nilai *throughput* pengunduhan pada waktu tertentu. Proses ini dilakukan dengan cara meminta *request* GET pada sebuah berkas yang berada pada *server*. Berdasarkan *request* tersebut, *server* akan mulai mengirimkan data berkas. Berkas yang digunakan pada uji coba ini adalah menggunakan berkas teks berformat .txt dengan ukuran berkas sejumlah 10 kB. Ukuran 10 kB ditentukan sebagai *sampling* ukuran paket yang terdapat pada ukuran *buffer* minimal pada protokol TCP. Pada uji coba performa pengunduhan ini yang dicatat adalah nilai *throughput rx (received bytes)* yang didapat oleh *client*. *Throughput rx* yang diambil adalah nilai *throughput rx controller* yang terdapat pada *port statistics*. Uji performa pengunduhan dibagi menjadi 2 (dua) yaitu uji performa pengunduhan tanpa *load switching* dan uji performa pengunduhan dengan *load switching*.

5.3.3.1. Uji Performa Pengunduhan tanpa Load Switching

Uji performa pengunduhan tanpa *load switching* dilakukan dengan melakukan *request* GET pada berkas dengan ukuran tertentu pada *server* freeshare.lp.if.its.ac.id. Berdasarkan *request* tersebut dihasilkan performa pengunduhan yang ditunjukkan pada gambar 5.25 berikut ini.



Gambar 5.25. Grafik Hasil Uji Performa Pengunduhan tanpa *Load Switching*

Berdasarkan data pada grafik hasil uji performa pengunduhan tanpa *load switching* diperoleh 2 (dua) titik dimana jaringan tidak dapat melakukan proses pengunduhan pada berkas yang telah ditentukan. Poin 1 dan 2 menunjukkan bahwa jaringan tidak dapat melakukan pengunduhan sama sekali. *Delay time* yang dihasilkan dari uji coba performa pengunduhan tanpa *load switching* ini ditunjukkan pada tabel 17 berikut ini.

Tabel 17. Tabel *Delay Time* pada Uji Performa Pengunduhan tanpa *Load Switching*

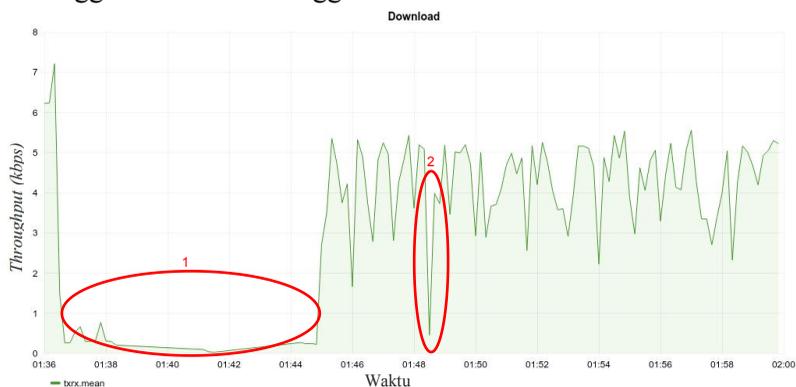
Nomor	<i>Delay Time (second)</i>
1	1010
2	270
total	1280

Berdasarkan tabel 13 diatas dapat diketahui bahwa waktu *delay time* pada uji performa pengunduhan tanpa *load switching* adalah 1280 detik.

5.3.3.2. Uji Performa Pengunduhan dengan *Load Switching* tanpa Waktu Tunggu

Sama seperti uji performa pengunduhan tanpa *load switching*, uji performa pengunduhan dengan *load switching*

tanpa waktu tunggu ini juga dilakukan dengan melakukan *request* GET pada berkas dengan ukuran tertentu pada *server* freeshare.lp.if.its.ac.id. Pada pengujian performa pengunduhan dengan *load switching* tanpa waktu tunggu ini dilakukan pemilihan parameter dan pengecekan *throughput* setiap iterasi. Gambar 5.26 berikut ini menunjukkan hasil uji coba pengujian performa pengunduhan dengan *load switching* tanpa menggunakan waktu tunggu.



Gambar 5.26. Grafik Hasil Uji Performa Pengunduhan dengan *Load Switching* tanpa Waktu Tunggu

Berdasarkan data pada grafik hasil uji performa pengunduhan dengan *load switching* tanpa waktu tunggu diperoleh 2 (dua) titik dimana jaringan tidak dapat melakukan proses pengunduhan pada berkas yang telah ditentukan dengan baik. Poin 1 dan 2 menunjukkan bahwa kecepatan unduh yang diterima oleh *client* sangat pelan yaitu dibawah 1 kbps.

Hasil *delay time* yang diperoleh dari uji performa pengunduhan dengan *load switching* tanpa waktu tunggu ditunjukkan pada tabel 18 berikut ini.

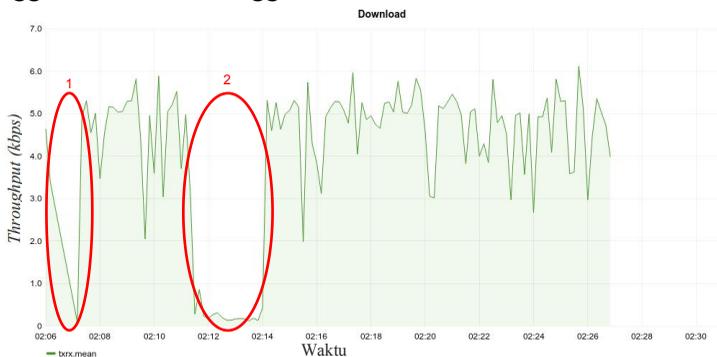
Tabel 18. Tabel *Delay Time* pada Uji Performa Pengunduhan dengan *Load Switching* tanpa Waktu Tunggu

Nomor	<i>Delay Time (second)</i>
1	570
2	20
total	590

Berdasarkan data pada table *delay time* diatas dapat diketahui bahwa uji performa pengunduhan dengan *load switching* tanpa waktu tunggu menghasilkan jumlah *delay time* 590 detik.

5.3.3.3. Uji Performa Pengunduhan dengan *Load Switching* menggunakan Waktu Tunggu

Pada uji performa pengunduhan dengan *load switching* tanpa waktu tunggu ini dilakukan dengan melakukan *request* GET pada berkas dengan ukuran tertentu pada server *freeshare.lp.if.its.ac.id*. Pada pengujian performa pengunduhan dengan *load switching* tanpa waktu tunggu ini dilakukan pemilihan parameter dan pengecekan *throughput* setiap 5 (lima) iterasi. Gambar 5.27 berikut ini menunjukkan hasil uji coba pengujian performa pengunduhan dengan *load switching* dengan menggunakan waktu tunggu.



Gambar 5.27. Grafik Hasil Uji Performa Pengunduhan dengan *Load Switching* menggunakan Waktu Tunggu

Berdasarkan data pada grafik hasil uji performa pengunduhan dengan *load switching* menggunakan waktu tunggu diperoleh 2 (dua) titik dimana jaringan tidak dapat melakukan proses pengunduhan pada berkas yang telah ditentukan dengan baik. Poin 1 dan 2 menunjukkan bahwa kecepatan unduh yang diterima oleh *client* sangat lambat yaitu dibawah 1 kbps.

Hasil *delay time* yang diperoleh dari uji performa pengunduhan dengan *load switching* menggunakan waktu tunggu ditunjukkan pada tabel 19 berikut ini.

Tabel 19. Tabel *Delay Time* pada Uji Performa Pengunduhan dengan *Load Switching* menggunakan Waktu Tunggu

Nomor	<i>Delay Time (second)</i>
1	70
2	170
total	240

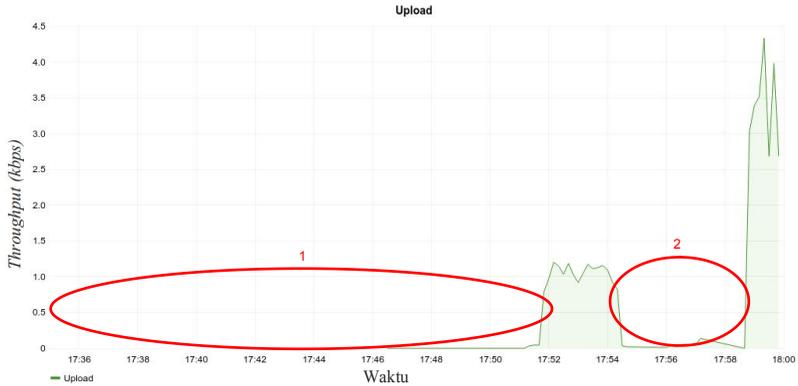
Berdasarkan data pada tabel *delay time* diatas dapat diketahui bahwa uji performa pengunduhan dengan *load switching* menggunakan waktu tunggu menghasilkan jumlah *delay time* 240 detik.

5.3.4. Uji Performa Pengunggahan

Uji performa pengunggahan merupakan uji performa yang dilakukan untuk mengetahui nilai *throughput* unggah pada waktu tertentu. Proses ini dilakukan dengan cara melakukan *request* PUT pada *server* dengan berkas tertentu. Berdasarkan *request* tersebut, maka *client* akan mengirimkan berkas kepada *server*. Pada uji coba performa pengunggahan ini yang dicatat adalah nilai *throughput tx* yang didapat oleh *client*. Uji performa pengunggahan dibagi menjadi 2 (dua) yaitu uji performa pengunggahan tanpa *load switching* dan uji performa pengunggahan dengan *load switching*.

5.3.4.1. Uji Performa Pengunggahan tanpa *Load Switching*

Uji performa pengunduhan tanpa *load switching* dilakukan dengan melakukan *request* PUT pada berkas dengan ukuran tertentu kepada *server* freeshare.lp.if.its.ac.id. Berdasarkan *request* tersebut dihasilkan performa pengunduhan yang ditunjukkan pada gambar 5.28 berikut ini.



Gambar 5.28. Grafik Hasil Uji Performa Pengunggahan Tanpa *Load Switching*

Berdasarkan data yang diperoleh dari grafik hasil uji performa pengunggahan tanpa *load switching* terdapat 2 (dua) titik dimana jaringan tidak dapat melakukan pengunggahan berkas pada *server* dengan baik. Poin 1 menunjukkan jaringan tidak dapat melakukan proses pengunggahan sama sekali. Poin 2 menunjukkan kecepatan unggah yang lambat, yaitu dibawah 1 kbps.

Waktu *delay time* yang dihasilkan dari uji performa pengunggahan tanpa *load switching* ditunjukkan pada tabel 20 berikut ini.

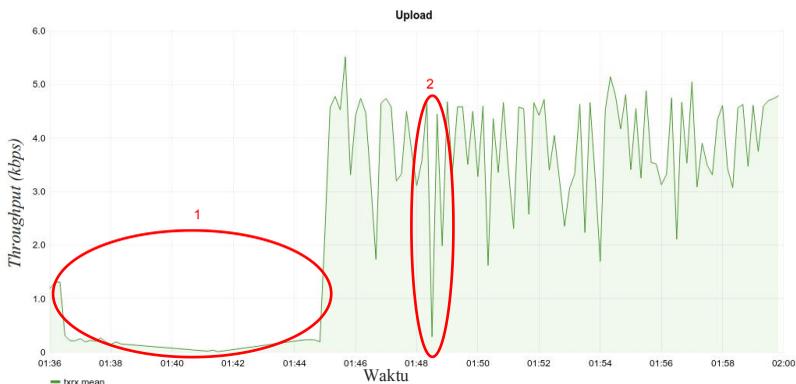
Tabel 20. Tabel *Delay Time* pada Uji Performa Pengunggahan tanpa *Load Switching*

Nomor	<i>Delay Time</i> (second)
1	1010
2	270
total	1280

Berdasarkan data pada tabel *delay time* diatas dapat diketahui bahwa uji performa pengunggahan tanpa *load switching* menghasilkan jumlah *delay time* 1280 detik.

5.3.4.2. Uji Performa Pengunggahan dengan *Load Switching* tanpa Waktu Tunggu

Uji performa pengunduhan dengan *load switching* tanpa waktu tunggu dilakukan dengan melakukan *request* PUT pada berkas dengan ukuran tertentu kepada *server* freeshare.lp.if.its.ac.id. Berdasarkan *request* tersebut dihasilkan performa pengunduhan yang ditunjukkan pada gambar 5.29 berikut ini.



Gambar 5.29. Grafik Hasil Uji Performa Pengunggahan dengan *Load Switching* tanpa Waktu Tunggu

Berdasarkan data yang diperoleh dari grafik hasil uji performa pengunggahan dengan *load switching* tanpa waktu tunggu terdapat 2 (dua) titik dimana jaringan tidak dapat

melakukan pengunggahan berkas pada *server* dengan baik. Poin 1 dan 2 menunjukkan kecepatan unggah yang lambat, yaitu dibawah 1 kbps.

Hasil *delay time* yang diperoleh dari uji performa pengunggahan dengan *load switching* tanpa waktu tunggu ditunjukkan pada tabel 21 berikut ini.

Tabel 21. Tabel *Delay Time* pada Uji Performa Pengunggahan dengan *Load Switching* tanpa Waktu Tunggu

Nomor	<i>Delay Time (second)</i>
1	590
2	20
total	610

Berdasarkan data pada tabel *delay time* diatas dapat diketahui bahwa uji performa pengunggahan dengan *load switching* tanpa waktu tunggu menghasilkan jumlah *delay time* 610 detik.

5.3.4.3. Uji Performa Pengunggahan dengan *load switching* menggunakan Waktu Tunggu

Uji performa pengunduhan dengan *load switching* menggunakan waktu tunggu dilakukan dengan melakukan *request* PUT pada berkas dengan ukuran tertentu kepada *server* freeshare.lp.if.its.ac.id. Berdasarkan *request* tersebut dihasilkan performa pengunduhan yang ditunjukkan pada gambar 5.30 berikut ini.



Gambar 5.30. Grafik Hasil Uji Performa Pengunggahan dengan *Load Switching* menggunakan Waktu Tunggu

Berdasarkan data yang diperoleh dari grafik hasil uji performa pengunggahan dengan *load switching* menggunakan waktu tunggu terdapat 2 (dua) titik dimana jaringan tidak dapat melakukan pengunggahan berkas pada *server* dengan baik. Poin 1 menunjukkan bahwa jaringan tidak dapat melakukan proses unduh sama sekali. Poin 2 menunjukkan kecepatan unggah yang lambat, yaitu dibawah 1 kbps.

Hasil *delay time* yang diperoleh dari uji performa pengunggahan dengan *load switching* menggunakan waktu tunggu ditunjukkan pada tabel 22 berikut ini.

Tabel 22. Tabel *Delay Time* pada Uji Performa Pengunggahan dengan *Load Switching* menggunakan Waktu Tunggu

Nomor	<i>Delay Time</i> (second)
1	70
2	220
total	290

Berdasarkan tabel *delay time* diatas dapat diketahui bahwa uji performa pengunggahan dengan *load switching* menggunakan waktu tunggu menghasilkan jumlah *delay time* 290 detik.

5.3.5. Ringkasan Uji Performa

Pada subbab ringkasan uji performa ini akan membahas mengenai perhitungan penurunan *delay time* dan penghitungan tingkat *reliability* pada uji performa yang telah dilakukan.

5.3.5.1. Penghitungan *Delay Time*

Pada penghitungan *delay time* ini akan dilakukan penghitungan rata-rata *delay time* dari hasil uji performa masing-masing metode yang dijelaskan pada tabel 23 sampai dengan tabel 25 dibawah ini

Tabel 23. Tabel Rata-rata *Delay Time* pada Uji Performa tanpa *Load Switching*

Uji Performa tanpa <i>Load Switching</i>	Jumlah <i>Delay Time</i> (detik)
Uji Performa <i>Network Reachable</i>	1106
Uji Performa <i>Round Trip Time</i>	1232
Uji Performa Pengunduhan	1280
Uji Performa Pengunggahan	1280
Rata-rata	1224,5

Tabel 24. Tabel Rata-rata *Delay Time* pada Uji Performa menggunakan *Load Switching* tanpa Waktu Tunggu

Uji Performa menggunakan <i>Load Switching</i> tanpa Waktu Tunggu	Jumlah <i>Delay Time</i> (detik)
Uji Performa <i>Network Reachable</i>	170
Uji Performa <i>Round Trip Time</i>	380
Uji Performa Pengunduhan	540
Uji Performa Pengunggahan	610
Rata-rata	425

Tabel 25. Tabel Rata-rata *Delay Time* pada Uji Performa menggunakan *Load Switching* tanpa Waktu Tunggu

Uji Performa menggunakan <i>Load Switching</i> dengan Waktu Tunggu	Jumlah <i>Delay Time</i> (detik)
Uji Performa <i>Network Reachable</i>	60
Uji Performa <i>Round Trip Time</i>	200
Uji Performa Pengunduhan	240
Uji Performa Pengunggahan	290
Rata-rata	197,5

5.3.5.2. Penghitungan Tingkat *Reliability*

Pada penghitungan tingkat *reliability* ini akan dilakukan penghitungan presentase peningkatan kualitas yang dihitung berdasarkan perubahan *delay time* pada setiap uji performa. Hasil penghitungan tingkat *reliability* ditunjukkan pada persamaan berikut ini.

Selisih *delay time* tanpa waktu tunggu = 1224,5 - 425 = 799,5

$$\text{reliability tanpa waktu tunggu} = \frac{799,5}{1224,5} \times 100 = 65,29 \%$$

Selisih *delay time* dengan waktu tunggu = 1224,5 - 197,5 = 1027

$$\text{reliability dengan waktu tunggu} = \frac{1027}{1224,5} \times 100 = 83,87 \%$$

[Halaman ini sengaja dikosongkan]

BAB VI PENUTUP

Pada bab terakhir ini akan dibahas mengenai kesimpulan yang diperoleh selama pengerjaan Tugas Akhir hingga selesai. Pada bab ini juga akan menjawab pertanyaan yang dijabarkan pada BAB I. Pembuatan Tugas Akhir ini pasti memiliki kelebihan dan kekurangan dari hasil yang telah dicapai dari pembuatan sistem. Untuk memperbaiki semua kekurangan dari sistem, dijelaskan pada subbab saran.

6.1. Kesimpulan

Berikut ini merupakan beberapa kesimpulan yang diperoleh dari proses pengerjaan Tugas Akhir ini:

1. Sistem dapat diimplementasikan dengan baik menggunakan *Software Defined Network* dengan *Controller OpenDaylight Helium SR-4 OSGi package* pada kendaraan mobil pribadi.
2. Sistem dapat melakukan deteksi *dynamic network* secara *realtime* dengan mengambil data *network reachable, throughput* dan *round trip time*.
3. Sistem dapat melakukan pemilihan koneksi terbaik ketika terjadi perubahan *bandwidth*.
4. Manajemen pada *client* dapat ditampilkan pada tampilan antarmuka *controller* dan *client* dapat terkoneksi pada sistem secara *wireless* menggunakan WiFi.
5. Sistem dapat melakukan manajemen *traffic* ketika berjalan dengan menerapkan sistem *flow* yang akan memberikan *policy* pada paket yang masuk dan keluar. (dituliskan uji coba ke berapa)

6. Sistem mampu memberikan laporan dan log jaringan menggunakan antarmuka *web* dan *terminal*.
7. Kualitas jaringan yang didapatkan menggunakan sistem *load switching* menggunakan *software defined network* memiliki pengurangan jumlah *delay time* rata-rata 799,5 detik apabila tanpa menggunakan waktu tunggu dan 1027 detik apabila menggunakan waktu tunggu. Meningkatkan *reliability* sebesar 65,29% apabila tanpa menggunakan waktu tunggu dan 83,87% apabila menggunakan waktu tunggu.

6.2. Saran

Adapun saran dari penulis yang diberikan dari kekurangan sistem yang ada, maupun pengembangan lebih lanjut yaitu:

1. Penambahan pengaruh kecepatan kendaraan sebagai parameter penentuan *flow*.
2. Penggunaan *machine learning* berdasarkan lokasi tertentu agar penentuan *flow* lebih cepat dan mengurangi redundansi data.
3. Penambahan fitur DHCP agar *client* yang terhubung ke sistem mendapatkan konfigurasi jaringan secara otomatis.
4. Deteksi konfigurasi perangkat diperlukan agar meminimalisasi interaksi manusia dalam melakukan konfigurasi sistem.
5. Penggunaan *hardware* yang seragam diperlukan agar pengambilan parameter pemilihan *flow* lebih baik.

DAFTAR PUSTAKA

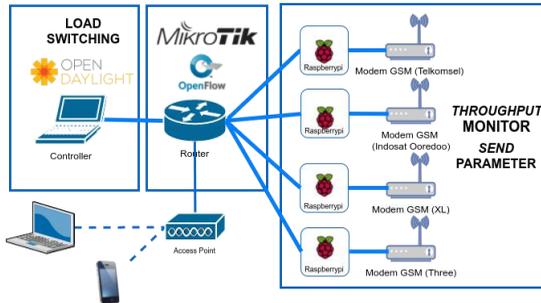
- [1] R. B'Far, “*Mobile Computing Principles*”, Cambridge: Cambridge University Press, 2005.
- [2] Kopparapu, Chandra, *load switching Servers, Firewalls, and Caches*, New York: New York, 2002.
- [3] C. Ellrod, “Load Balancing – Round Robin,” Citrix, 3 September 2010. [Online]. Available: <https://www.citrix.com/blogs/2010/09/03/load-balancing-round-robin/>. [Diakses 6 Januari 2017].
- [4] C. Ellrod, “Load Balancing – Least Connections,” Citrix, 2 September 2010. [Online]. Available: <https://www.citrix.com/blogs/2010/09/02/load-balancing-least-connections/>. [Diakses 6 Januari 2017].
- [5] C. Ellrod, “Load Balancing – Hash Method,” Citrix, 4 September 2010. [Online]. Available: <https://www.citrix.com/blogs/2010/09/04/load-balancing-hash-method/>. [Diakses 6 Januari 2017].
- [6] W. Stallings, “*Wireless Connections and Networking Second Edition*”, New Jersey: Pearson Education, 2005.
- [7] “*Software-Defined Networking (SDN) Definition*” , Open Networking Foundation. [online]. Available: <https://www.opennetworking.org/sdn-resources/sdn-definition>. [diakses 6 Januari 2017].
- [8] “*OpenDaylight User Guide*”, Linux Foundation. [Online]. Available: <https://www.opendaylight.org/sites/opendaylight/files/bk-user-guide.pdf>. [diakses 6 Januari 2017].
- [9] “*OpenDaylight Helium Application Developers’ Tutorial*” , SDN Hub. [Online]. Available: <http://sdnhub.org/tutorials/opendaylight-helium/>. [diakses 6 Januari 2017].
- [10] Fielding, Roy Thomas. “*Architectural Styles and the Design of Network-based Software Architectures*”. University of

California, Irvine: 2000. [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. [diakses 6 Januari 2017].

- [11] "*Wireless Local Area Network (WLAN)*", Techopedia. [Online]. Available: <https://www.techopedia.com/definition/5107/wireless-local-area-network-wlan>. [diakses 6 Januari 2017].
- [12] "OpenFlow", The McKeown Research Group. Stanford University. [Online]. Available : <http://yuba.stanford.edu/cs244/wiki/index.php/Overview>. [diakses 6 Januari 2017].
- [13] "OpenFlow", Modeling of Security and Systems (MOSES) Research Group. University of Illinois. [Online]. Available : <https://s3f.iti.illinois.edu/usrman/openflow.html>. [diakses 6 Januari 2017].

LAMPIRAN

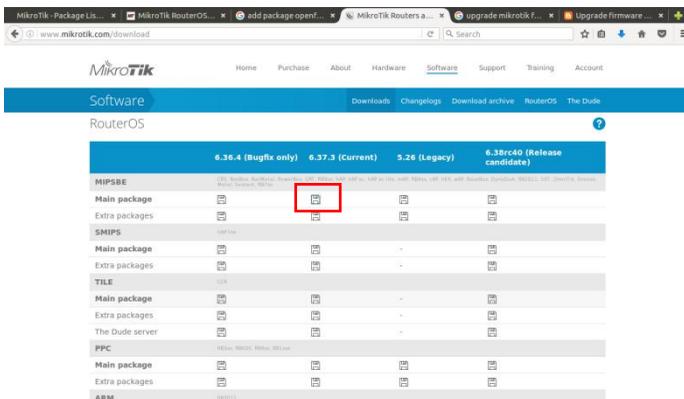
Bagian ini merupakan lampiran dari dokumen sebagai pelengkap buku Tugas Akhir dimana akan diberikan langkah-langkah konfigurasi sistem dan potongan kode sumber program dari fungsi-fungsi yang digunakan.



A. Langkah-langkah Konfigurasi Mikrotik dengan OpenFlow

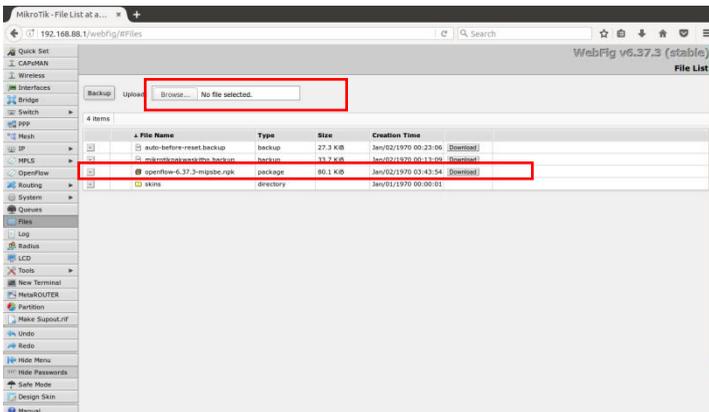
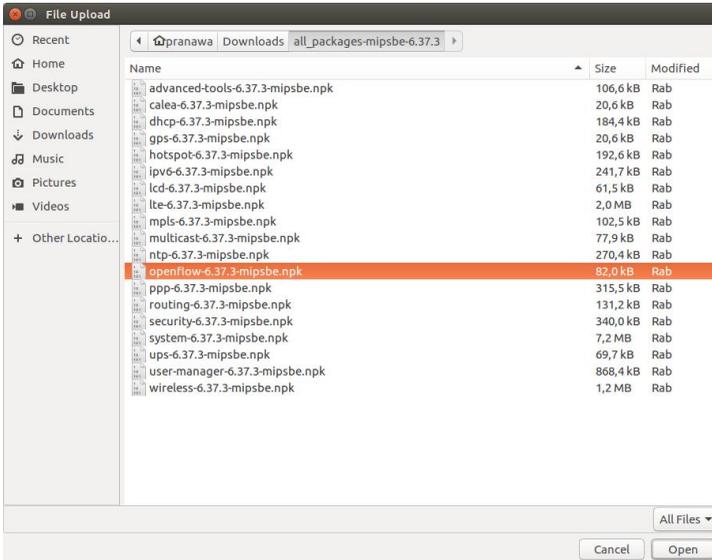
1. *Download* Paket OpenFlow

Download paket OpenFlow Mikrotik pada alamat <http://www.mikrotik.com/download>



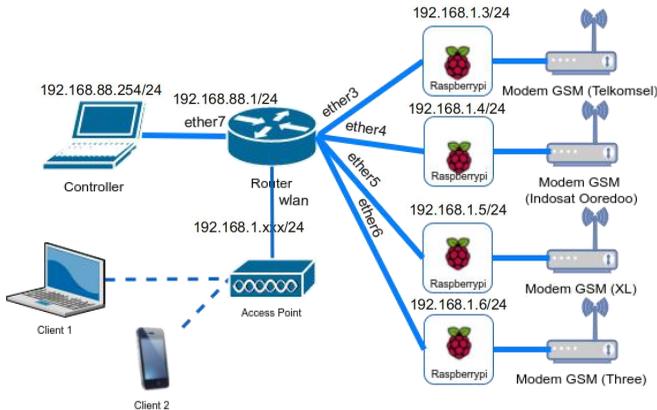
2. Install Paket OpenFlow

Extract kemudian *Install* paket OpenFlow kedalam *router* melalui menu *files*.



Kemudian *reboot* router

3. Konfigurasi Alamat IP



4. Konfigurasi Interface

Lakukan konfigurasi pada *router* dengan melakukan *setting master-port = none*.

```
[admin@MikroTik] /interface ethernet> set ether3,ether4,ether5,ether6 master-port=none
[admin@MikroTik] /interface ethernet> print
Flags: X - disabled, R - running, S - slave
```

#	NAME	MTU	MAC-ADDRESS	ARP	MASTER-PORT	SWITCH
0	ether1	1500	E4:8D:8C:AF:43:2E	enabled	none	switch1
1	S ether2...	1500	E4:8D:8C:AF:43:2F	enabled	none	switch1
2	ether3	1500	E4:8D:8C:AF:43:30	enabled	none	switch1
3	ether4	1500	E4:8D:8C:AF:43:31	enabled	none	switch1
4	ether5	1500	E4:8D:8C:AF:43:32	enabled	none	switch1
5	ether6	1500	E4:8D:8C:AF:43:33	enabled	none	switch1
6	S ether7	1500	E4:8D:8C:AF:43:34	enabled	ether2-master	switch1
7	S ether8	1500	E4:8D:8C:AF:43:35	enabled	ether2-master	switch1
8	S sfp1	1500	E4:8D:8C:AF:43:36	enabled	ether2-master	switch1

5. Konfigurasi OpenFlow

```
[admin@MikroTik] > OpenFlow add
name=ofswitch1 controllers=192.168.88.254
[admin@MikroTik] > OpenFlow enable ofswitch1
[admin@MikroTik] > OpenFlow port add
switch=ofswitch1 interface=ether3
[admin@MikroTik] > OpenFlow port add
```

```

switch=ofswitch1 interface=ether4
[admin@MikroTik] > OpenFlow port add
switch=ofswitch1 interface=ether5
[admin@MikroTik] > OpenFlow port add
switch=ofswitch1 interface=ether6
[admin@MikroTik] > OpenFlow port add
switch=ofswitch1 interface=wlan
[admin@MikroTik] > OpenFlow port enable
numbers=0
[admin@MikroTik] > OpenFlow port enable
numbers=1
[admin@MikroTik] > OpenFlow port enable
numbers=2
[admin@MikroTik] > OpenFlow port enable
numbers=3
[admin@MikroTik] > OpenFlow port enable
numbers=4

```

B. Langkah-langkah Konfigurasi *Controller* dengan OpenDaylight

1. *Install Java OpenJDK*

```
$ sudo apt-get install openjdk-8-jre
```

2. *Download Paket Controller OpenDaylight-OSGi*

```

$ wget
https://nexus.OpenDaylight.org/content/repositories/OpenDaylight.release/org/OpenDaylight/controller/distribution.OpenDaylight/0.1.6-Helium-SR4/distribution.OpenDaylight-0.1.6-Helium-SR4-osgipackage.zip

```

3. *Extract Paket Controller OpenDaylight-OSGI*

```
$ unzip distribution.OpenDaylight-0.1.6-
Helium-SR4-osgipackage.zip
```

4. *Setting Environment JAVA_HOME*

```
$ export JAVA_HOME=/usr/lib/jvm/java-xx-
openjdk-yy
```

5. *Eksekusi Aplikasi Controller OpenDaylight-OSGI*

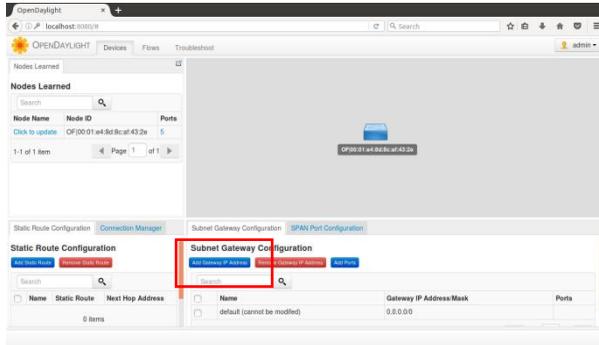
```
$ sudo ./run.sh
```

```
l.java:95] [bundlefile:na]
   at org.opendaylight.controller.sal.binding.impl.forward.DomForwardedBindingBrokerImpl.tryToDeployDomForwarder(DomForwardedBindingBroker
Impl.java:114) [bundlefile:na]
   at org.opendaylight.controller.sal.binding.impl.forward.DomForwardedBindingBrokerImpl$DOMMountPointForwardingManager.onMountPointCreate
d(DomForwardedBindingBrokerImpl.java:137) [bundlefile:na]
   at org.opendaylight.controller.md.sal.dom.broker.impl.mount.DOMMountPointServiceImpl.notifyMountCreated(DOMMountPointServiceImpl.java:4
0) [bundlefile:na]
   at org.opendaylight.controller.md.sal.dom.broker.impl.mount.DOMMountPointServiceImpl.registerMountPoint(DOMMountPointServiceImpl.java:7
0) [bundlefile:na]
   at org.opendaylight.controller.md.sal.dom.broker.impl.mount.DOMMountPointServiceImpl$DOMMountPointBuilderImpl.register(DOMMountPointSer
viceImpl.java:110) [bundlefile:na]
   at org.opendaylight.controller.sal.connect.netconf.sal.NetconfDeviceSalProviderMountInstance.onDeviceConnected(NetconfDeviceSalProvide
r.java:127) [bundlefile:na]
   at org.opendaylight.controller.sal.connect.netconf.sal.NetconfDeviceSalFacade.onDeviceConnected(NetconfDeviceSalFacade.java:93) [bundle
file:na]
   at org.opendaylight.controller.sal.connect.netconf.sal.NetconfDeviceSalFacade.onDeviceConnected(NetconfDeviceSalFacade.java:38) [bundle
file:na]
   at org.opendaylight.controller.sal.connect.netconf.NetconfDevice.handleSalInitializationSuccess(NetconfDevice.java:126) [bundlefile:na]
   at org.opendaylight.controller.sal.connect.netconf.NetconfDevice.access$800(NetconfDevice.java:54) [bundlefile:na]
   at org.opendaylight.controller.sal.connect.netconf.NetconfDevice$RecursiveSchemaSetup$1.onSuccess(NetconfDevice.java:325) [bundlefile:na]
a)   at org.opendaylight.controller.sal.connect.netconf.NetconfDevice$RecursiveSchemaSetup$1.onSuccess(NetconfDevice.java:326) [bundlefile:na]
a)   at com.google.common.util.concurrent.Futures$4.run(Futures.java:1149) [bundlefile:na]
   at com.google.common.util.concurrent.MoreExecutors$SameThreadExecutorService.execute(MoreExecutors.java:293) [bundlefile:na]
   at com.google.common.util.concurrent.ExecutionList$RunnableExecutorPair.execute(ExecutionList.java:150) [bundlefile:na]
   at com.google.common.util.concurrent.ExecutionList.add(ExecutionList.java:106) [bundlefile:na]
   at com.google.common.util.concurrent.AbstractFuture.addListener(AbstractFuture.java:370) [bundlefile:na]
   at com.google.common.util.concurrent.ForwardingListenableFuture.addListener(ForwardingListenableFuture.java:47) [bundlefile:na]
   at com.google.common.util.concurrent.Futures.addCallback(Futures.java:1192) [bundlefile:na]
   at com.google.common.util.concurrent.Futures.addCallback(Futures.java:1088) [bundlefile:na]
   at org.opendaylight.controller.sal.connect.netconf.NetconfDevice$RecursiveSchemaSetup.setupSchema(NetconfDevice.java:349) [bundlefile:na]
a)   at org.opendaylight.controller.sal.connect.netconf.NetconfDevice$RecursiveSchemaSetup.run(NetconfDevice.java:303) [bundlefile:na]
   at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511) [na:1.8.0_111]
   at java.util.concurrent.FutureTask.run(FutureTask.java:266) [na:1.8.0_111]
   at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142) [na:1.8.0_111]
   at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617) [na:1.8.0_111]
   at java.lang.Thread.run(Thread.java:745) [na:1.8.0_111]
o.s.d.s.c.n.c.netconf.NetconfDevice - RemoteDevice[controller-config]:
Netconf connector initialized successfully
osgi> █
```

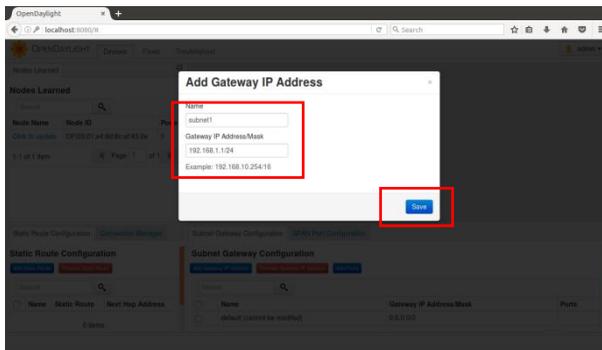
Apabila telah keluar status *netconf connector initialized successfully*, maka aplikasi *controller* telah berhasil dipasang dan dijalankan.

6. *Konfigurasi Aplikasi Controller melalui Browser*

Akses *interface controller* melalui *browser* dengan *port 8080* kemudian masukkan *username admin* *password admin*. Tambahkan *subnet gateway address* sesuai dengan jaringan yang dipakai.



Kemudian simpan alamat IP dan simpan konfigurasi OpenDaylight *controller* agar tersimpan permanen.



C. Langkah-langkah Konfigurasi Basis Data InfluxDB dan *Network Monitoring Grafana*

1. Tambah *Repository* InfluxDB

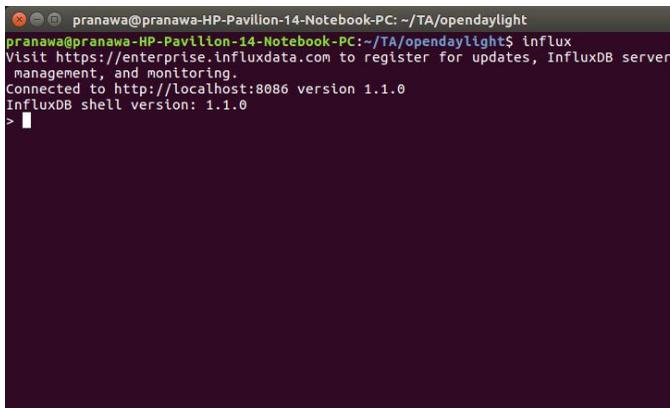
```
$ curl -sL
https://repos.influxdata.com/influxdb.key | sudo
apt-key add -
$ source /etc/lsb-release
$ echo "deb
https://repos.influxdata.com/${DISTRIB_ID,,}
${DISTRIB_CODENAME} stable" | sudo tee
/etc/apt/sources.list.d/influxdb.list
```

2. *Install* InfluxDB

```
$ sudo apt-get update && sudo apt-get install
influxdb
```

3. Jalankan *Service* InfluxDB

```
$ sudo service influxdb start
$ influx
```



```
pranawa@pranawa-HP-Pavllion-14-Notebook-PC: ~/TA/opendaylight
pranawa@pranawa-HP-Pavllion-14-Notebook-PC:~/TA/opendaylight$ influx
Visit https://enterprise.influxdata.com to register for updates, InfluxDB server
management, and monitoring.
Connected to http://localhost:8086 version 1.1.0
InfluxDB shell version: 1.1.0
>
```

4. Buat *Database* controllerdb dan Tabel portstats

```
$ influx
```

```
$ create database controllerdb
$ use database controllerdb
$ insert portstats,host=nodeX tx=0,rx=0
```

5. Tambah *Repository* Grafana

```
$ echo "deb
https://packagecloud.io/grafana/stable/debian/
wheezy main" | sudo tee
/etc/apt/sources.list.d/grafana.list
$ curl https://packagecloud.io/gpg.key | sudo
apt-key add -
```

6. *Install Grafana*

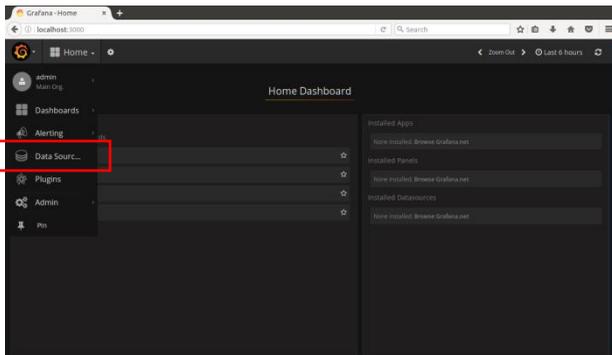
```
$ sudo apt-get update && sudo apt-get install
grafana
```

7. Jalankan *Service Grafana*

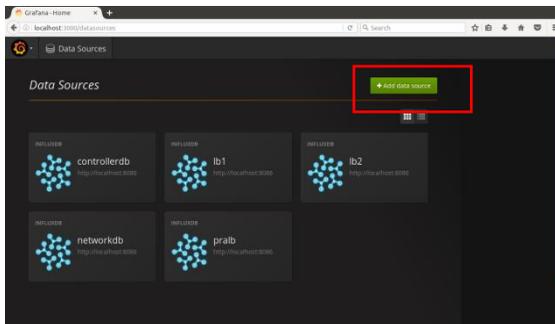
```
$ sudo service grafana-server start
```

8. *Buat Dashboard Network Monitoring*

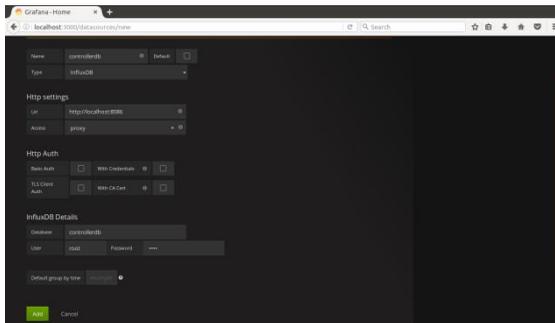
Akses *interface* Grafana melalui *browser* dengan *port 3000*. Masukkan *username admin* *password admin*. Klik *Data Source*



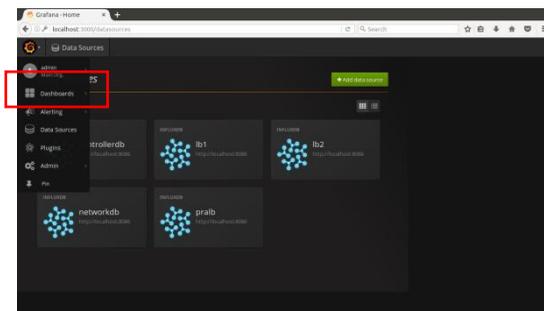
Klik *Add data source*



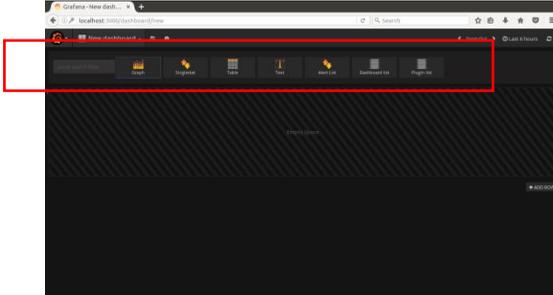
Isi beberapa parameter pada kolom *name*, *type*, *url*, *access*, *database*, *user*, *password*. Kemudian klik *add*.



Setelah *data source* ditambahkan, buat *dashboard* klik **+new**



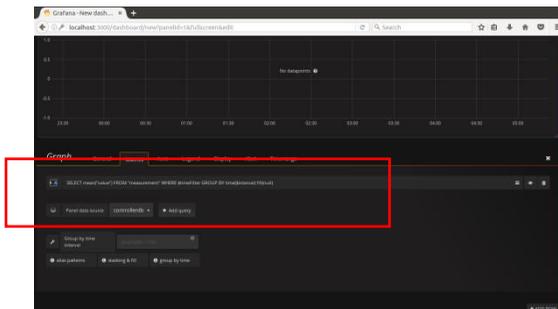
Pilih antarmuka yang ingin ditampilkan, pada konfigurasi ini dipilih *graph*.



Klik **judul** pada grafik, kemudian klik *Edit*.



Isi parameter *panel data source* dengan *data source* yang telah ditambahkan dan sesuaikan *query* untuk menampilkan grafik.



D. Konfigurasi *Host Router*

1. *Install wvdial*

```
$ sudo apt-get update && sudo apt-get install grafana
```

2. Konfigurasi wvdial

```
$ sudo wvdialconf
```

Lakukan konfigurasi wvdial untuk masing-masing ISP.

3. Konfigurasi *Forwarding*

```
$ sudo iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
```

```
$ sudo iptables -A FORWARD -i ppp0 -o eth0 -m state --state RELATED,ESTABLISHED -j ACCEPT
```

```
$ sudo iptables -A FORWARD -i eth0 -o ppp0 -j ACCEPT
```

E. Potongan Kode Program *Load Switching* tanpa Waktu Tunggu dengan REST API

Langkah pertama adalah melakukan inialisasi *gateway mac address*. Normalnya, *gateway mac address* adalah *mac address interface mesin controller* yang terhubung dengan *router*.

Langkah kedua adalah mengambil data *host* yang terhubung pada OpenFlow *router* beserta *node ID* OpenFlow *router* tersebut. Kode program ini ditunjukkan pada kode sumber 1 berikut ini.

```
h = httpLib2.Http(".cache")
h.add_credentials("admin", "admin")
resp, content=h.request("http://localhost:8080/controller/nb/v2/hosttracker/default/hosts/active")
```

```
)
jsonData = json.loads(content)
hostData = jsonData['hostConfig']
```

Kode Sumber 1. Kode Sumber Pengambilan Data *Host* pada OpenFlow Router.

Langkah ketiga adalah menginisialisasi *master node* atau *host router* yang dijadikan koneksi utama. Kode program ini ditunjukkan pada kode sumber 2 berikut ini.

```
os.system('curl -u admin:admin -H "Content-type: application/json" -X PUT -d "{\\"installInHw\\":\\"true\\"",\\"name\\":\\"flowx\\"",\\"node\\":{\\"id\\":\\"'+nodeID+'\\",\\"type\\":\\"OF\\"},\\"etherType\\":\\"0x800\\"",\\"priority\\":\\"500\\"",\\"d1Dst\\":\\"'+gateway+'\\"",\\"actions\\":[\\"SET_DL_DST='+macMasterNode+'\\",\\"OUTPUT=N\\"]}" "http://localhost:8080/controller/nb/v2/flowprogrammer/default/node/OF/'+nodeID+'/staticFlow/flowx"')
```

Kode Sumber 2. Kode Sumber Inisialisasi *Master Node*.

Flowx adalah nama *flow* yang digunakan, **nodeID** adalah ID Unik OpenFlow *router* yang didapat setelah pengambilan data *host*, **gateway** adalah *gateway mac address*. **macMasterNode** adalah *mac address* dari *master node* yang digunakan, **N** adalah nomor *port* pada *master node*.

Langkah keempat yaitu dilakukan pengambilan data *throughput* yang terdapat pada tabel *port controller*. Kode program ini ditunjukkan pada kode sumber 3 berikut ini.

```
resp, content=h.request("http://127.0.0.1:8080/controller/nb/v2/statistics/default/port/")
```

Kode Sumber 3. Kode Sumber Pengambilan Data *Throughput* pada Tabel *Port Controller*.

Langkah kelima adalah memasukan data *throughput* diatas kedalam *database* InfluxDB. Kode program ini ditunjukkan pada kode sumber 4 berikut ini.

```
influxdata="host=%s tx=%s,rx=%s" % (nodename,
txdata, rxdata)
os.system("curl -XPOST
'http://%s:%s/write?db=%s' --data-binary
'portstats,%s'" % (host, port, dbname,
influxdata))
```

Kode Sumber 4, Kode Sumber Memasukkan Data *Throughput* kedalam *Database*.

Langkah keenam yaitu melakukan pengambilan data *throughput* terbaik pada *database*. Kode program ini ditunjukkan pada kode sumber 5 berikut ini.

```
query='select max(rx) as maxrx, host from
portstats where time > now() - 1s'
client=InfluxDBClient(host,port,user,password,db
name)
result = client.query(query)
arrayList=list(result.get_points(measurement='po
rtstats'))
maxrx=arrayList[0]['maxrx']
node=arrayList[0]['host']
```

Kode Sumber 5. Kode Sumber Pengambilan Data *Throughput* Terbaik pada *Database* tanpa Waktu Tunggu.

Langkah ketujuh yaitu melakukan perbandingan dan pembuatan *flow* berdasarkan jumlah *throughput* terbaik. Kode program ini ditunjukkan pada kode sumber 6 berikut ini.

```
if maxData[0]=='nodeName' and maxData[1]>0:
    os.system('curl -u admin:admin -H
"Content-type: application/json" -X PUT -d
"{\\"installInHw\\":\\"true\\"},\\"name\\":\\"
flowx\\",\\"node\\":{\\"id\\":\\"'+nodeID+'
```

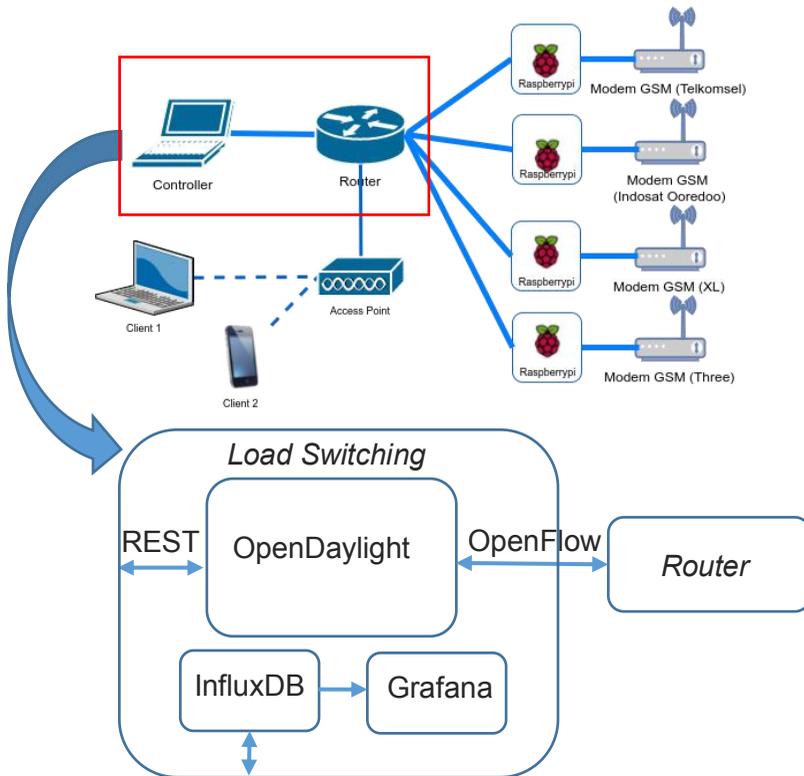
```

\\",\\"type\\":\\"OF\\"},\\"etherType\\":\\"
0x800\\",\\"priority\\":\\"500\\",\\"d1Dst\\":
\\"'+gateway+'\\",\\"actions\\":[\\"SET_DL
_DST='+macMasterNode+'\\",\\"OUTPUT=N\\"}]
"http://localhost:8080/controller/nb/v2/flow
programmer/default/node/OF/' +nodeID+' /static
Flow/flowx"')

```

Kode Sumber 6. Kode Sumber Perbandingan dan Pembuatan *Flow*. *nodeName* adalah nama *host router* yang digunakan sebagai *gateway* ke *internet*.

F. Gambaran Umum Cara Kerja Aplikasi



G. Potongan Kode Program *Load Switching* dengan Waktu Tunggu dengan REST API

Secara umum, kode program *load switching* dengan waktu tunggu memiliki kesamaan dengan kode program *load switching* tanpa waktu tunggu. Tetapi kode program *load switching* dengan waktu tunggu terdapat perubahan penambahan *counter* dan perbedaan kode program pengambilan *throughput* pada tabel *port controller*. Kode program ini ditunjukkan pada kode sumber 7 berikut ini.

```
query='select max(rx) as maxrx, host from
portstats where time > now() - 5s'
client=InfluxDBClient(host,port,user,password,db
name)
result = client.query(query)
arrayList=list(result.get_points(measurement='po
rtstats'))
maxrx=arrayList[0]['maxrx']
node=arrayList[0]['host']
```

Kode Sumber 7. Kode Sumber Pengambilan Data *Throughput* Terbaik pada *Database* dengan Waktu Tunggu.

H. Potongan Kode Program Pengiriman Parameter pada *Host Router*

Kode program pengiriman parameter pada *host router* terbagi menjadi 3 (tiga) bagian, yaitu pengambilan *round trip time*, pengambilan *throughput*, serta penghitungan dan pengiriman *parameter*.

Kode program pengambilan *round trip time* ditunjukkan pada kode sumber 8 berikut ini.

```
os.system("ping -c 1 8.8.8.8 | awk 'BEGIN
{FS="+ "[=] | [ ]"+" } {print $10}' | head -n 2 |
tail -n 1 > pingout.txt ")
with open('pingout.txt','r') as f:
    data=f.read()
```

```

if data=='':
    data=0
return float(data)

```

Kode Sumber 8. Kode Sumber Pengambilan *Round Trip Time*.

Kode program pengambilan *throughput modem* ditunjukkan pada kode sumber 9 berikut ini.

```

with open('/sys/class/net/'+iface+'/statistics/'
+ t + '_bytes', 'r') as f:
    data=f.read()
return float(data)

```

Kode Sumber 9. Kode Program Pengambilan *Throughput Modem*.

iface merupakan *interface* yang digunakan *modem* untuk koneksi ke *internet*. Normalnya, *interface* ini menggunakan **ppp0**. **t** merupakan jenis *throughput* yang diambil (tx atau rx).

Kode program penghitungan dan pengiriman parameter ditunjukkan pada kode sumber 10 berikut ini.

```

param=(rbps+tbps)/rtt
os.system('ping -s '+str(param)+' -c 1
192.168.1.x')

```

Kode Sumber 10. Kode program penghitungan dan pengiriman parameter.

rbps merupakan *throughput rx* dalam *bit per seconds*. **tx** merupakan *throughput tx* dalam *bit per seconds*. **rtt** merupakan *round trip time*, **x** merupakan alamat IP *host router* yang menjadi tetangganya.

BIODATA PENULIS



Yoga Bayu Aji Pranawa, lahir pada tanggal 20 April 1995 di Klaten, Jawa Tengah. Penulis merupakan seorang mahasiswa yang sedang menempuh studi di Jurusan Teknik Informatika Institut Teknologi Sepuluh Nopember Surabaya. Penulis memiliki hobi bermain, editing, dan producing musik terutama musik komersil. Penulis juga merupakan staff divisi kementerian komunikasi dan informasi Badan Eksekutif Mahasiswa (BEM) Insitut Teknologi Sepuluh Nopember periode 2014-2015. Aktivitas di dunia Teknologi Informasi tidak hanya dikembangkan di lingkup perkuliahan saja. Penulis mengikuti beberapa komunitas Tekonogi Informasi antara lain, Indonesian Backtrack Team, VMware *client group* Asean, Cisco *network engineer* Indonesia. Beberapa pelatihan di bidang Teknologi informasi pernah diikuti, diantaranya *workshop network security* ICROSS, *workshop Cisco certification*, *workshop Cisco for corporate*, *Virtualization*, *workshop* desain dan implementasi *clustering* Windows *server*. Beberapa sertifikasi dan penghargaan internasional pernah diraih, diantaranya Cisco *Certified Network Associate* (CCNA), Microsoft *Technology Associate* (MTA), VMware vExpert 2015, dan VMware vExpert 2016. Penulis juga pernah menjadi pembicara dalam komunitas Indonesian Backtrack Team regional Surabaya dalam hal *server virtualization* dan pernah memegang proyek berskala internasional bersama dengan Neco Asia Pacific.

[Halaman ini sengaja dikosongkan]

[Halaman ini sengaja dikosongkan]