

**UNIVERSITAS INDONESIA**

# ANALISIS KINERJA PADA MEKANISME SKEMA *FAST HANDOFF* DALAM JARINGAN *WI-FI* BERBASIS *OPENFLOW*

**SKRIPSI**

**ISYANA PARAMITHA ISKANDARPUTRI**

**1106014135**

**FAKULTAS TEKNIK**

**PROGRAM STUDI TEKNIK KOMPUTER**

**DEPOK**

JANUARI 2015



**UNIVERSITAS INDONESIA**

# ANALISIS KINERJA PADA MEKANISME SKEMA *FAST HANDOFF* DALAM JARINGAN *WI-FI* BERBASIS *OPENFLOW*

**SKRIPSI**

**Diajukan sebagai salah satu syarat untuk memperoleh gelar Sarjana Teknik**

**ISYANA PARAMITHA ISKANDARPUTRI**

**1106014135**

**FAKULTAS TEKNIK**

**PROGRAM STUDI TEKNIK KOMPUTER**

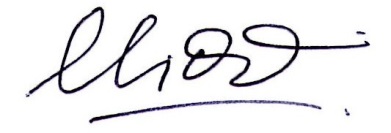
**DEPOK**

JANUARI 2015

**HALAMAN PERNYATAAN ORISINALITAS**

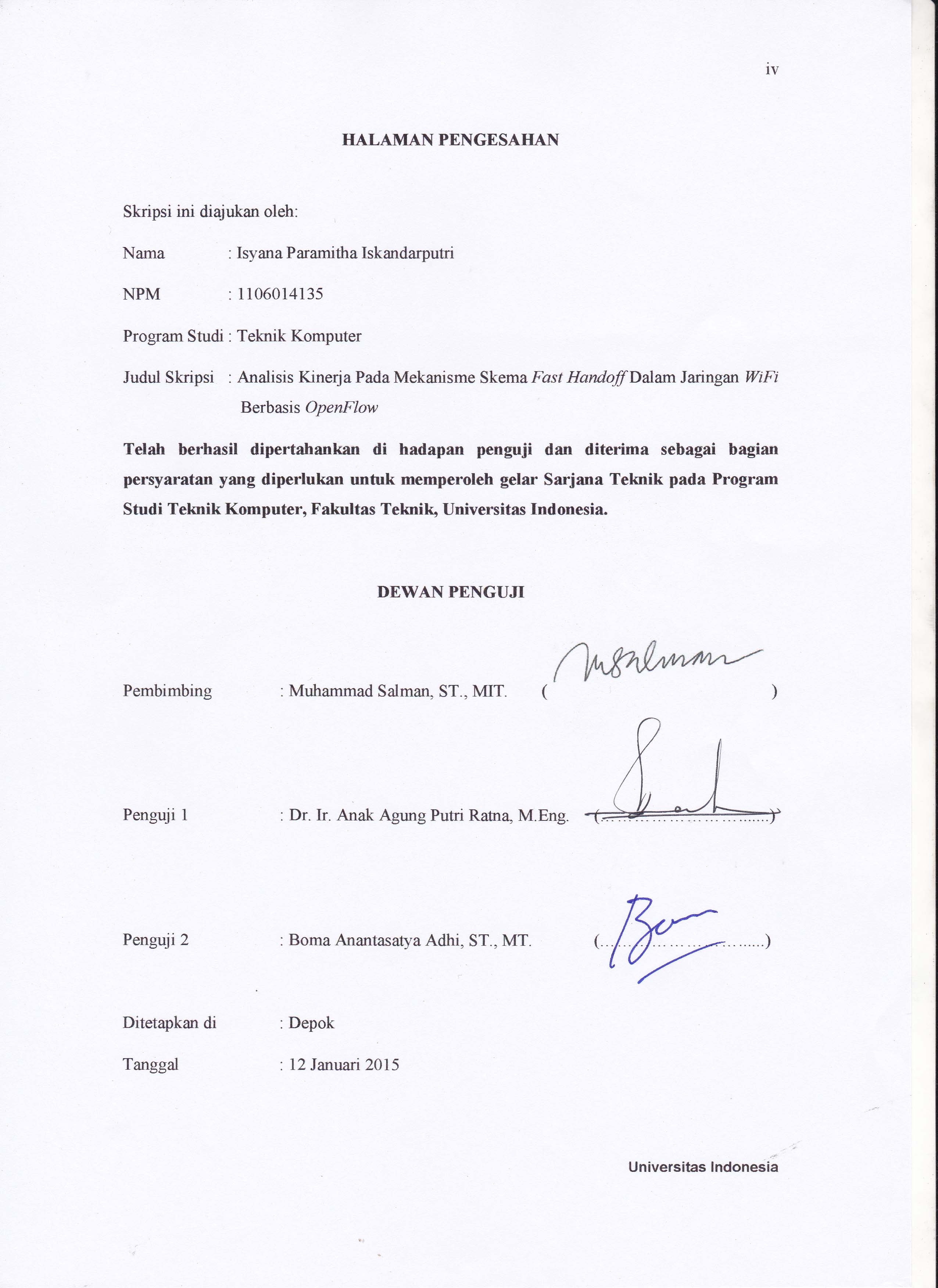
**Skripsi ini adalah hasil karya saya sendiri, dan semua sumber baik yang dikutip maupun dirujuk telah saya nyatakan dengan benar.**

**Nama : Isyana Paramitha Iskandarputri**

**NPM : 1106014135**

**Tanda Tangan :**

**Tanggal : 12 Januari 2015**

****

# HALAMAN PENGESAHAN

Skripsi ini diajukan oleh:

Nama : Isyana Paramitha Iskandarputri

NPM : 1106014135

Program Studi : Teknik Komputer

Judul Skripsi : Analisis Kinerja Pada Mekanisme Skema *Fast Handoff* Dalam Jaringan *Wi-Fi* Berbasis *OpenFlow*

**Telah berhasil dipertahankan di hadapan penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Teknik pada Program Studi Teknik Komputer, Fakultas Teknik, Universitas Indonesia.**

**DEWAN PENGUJI**

Pembimbing : Muhammad Salman, ST., MIT. (……………………)

Penguji 1 : Dr. Ir. Anak Agung Putri Ratna, M.Eng. (……………………)

Penguji 2 : Boma Anantasatya Adhi, ST., MT. (……………………)

Ditetapkan di : Depok

**KATA PENGANTAR**

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa sebab atas segala rahmat dan karunia-Nya, penulis dapat menyelesaikan seminar ini dengan baik. Penulis menyadari bahwa skripsi ini tidak dapat diselesaikan tanpa bantuan dari banyak pihak. Oleh karena itu, penulis mengucapkan terima kasih kepada:

1. Bapak Muhammad Salman, S.T., MIT., selaku pembimbing yang telah mengarahkan serta memberi berbagai bantuan dalam penyusunan skripsi ini;
2. Orang tua dan keluarga penulis serta Kenneth Setiawan yang telah memberikan dukungan berupa moril maupun materiil;
3. Kak Ardiansyah, S.T., M.Eng, Kak Alvin Prayuda, S.T, Pak I Gde Dharma Nugraha, S.T., M.T., dan Kak Boma Anantasatya Adhi, S.T., M.T., yang telah membantu dan memberi saran dalam penulisan seminar ini;
4. Farina Warapsari, Anindita Putri H., Keluarga Sambel Sedep (Fitri Amelina, Gea Rexy, Kemal Ardiansyah, Rinaldi Adrian, Waskitha W. Galih, Deni Duwi) dan Keluarga Oma Mpuy (Dwi Hartati, Dinar Ayu, Maulidha Syaila, Hafizh Haidar, Muhamad Sean, Mahardianto Yudha, M. Rizqi Kautsar, Dhika Pratama, M. Mazarino Zhafir), Jodhi Adhikaprana, Gandhi Arssegi, Ihsan Ibrahim atas dorongan semangatnya;
5. Rekan sesama asisten laboratorium Jaringan Komputer 2011 dan 2012 (Adhyatma Abbas, Sanadhi Sutandi, Darien Jonathan, Emily Lomempow, Ozy Djohan, Suryo Satrio, Martin Dominikus, Akbar Sahata, Fakhri Aunurrahim);
6. Teman-teman Teknik Komputer dan Teknik Elektro 2011 yang sudah menemani dan secara tidak langsung memberikan dukungan moral agar terselesainya skripsi ini dan seluruh Keluarga Departemen Teknik Elektro.

Penulis berharap agar Tuhan YME akan membalas kebaikan semua pihak yang telah membantu dan semoga skripsi ini dapat bermanfaat bagi pengembangan teknologi dan ilmu pengetahuan.

Depok, Januari 2015

# 

# HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI SKRIPSI UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini :

Nama : Isyana Paramitha

NPM : 1106014135

Program Studi : Teknik Komputer

Departemen : Teknik Elektro

Fakultas : Teknik

Jenis Karya : Skripsi

Demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif (*Non-exclusive Royalty Free Right*)** atas karya ilmiah saya yang berjudul:

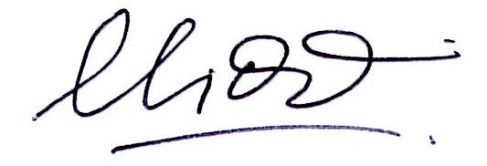
“Analisis Kinerja Pada Mekanisme Skema *Fast Handoff* Dalam Jaringan *Wi-Fi* Berbasis *OpenFlow*”

Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalih media/formatkan, mengelola dalam bentuk pengkalan data (*database*), merawat, dan mempublikasikan tugas akhir saya selama tetap mencamtumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenar-benarnya.

Dibuat di: Depok

Pada tanggal: 12 Januari 2015

****Yang menyatakan

(Isyana Paramitha Iskandarputri)

ABSTRAK

|  |  |  |
| --- | --- | --- |
| Nama | : | Isyana Paramitha |
| Program Studi | : | Teknik Komputer |
| Judul | : | Analisis Kinerja Pada Mekanisme Skema *Fast Handoff* Dalam Jaringan *WiFi* Berbasis *OpenFlow* |

Arah pengembangan jaringan saat ini adalah pemanfaatan *open source* mengingat terbatasnya sumber untuk pengembangan teknologi jaringan baik untuk keperluan riset berbasis akademis ataupun *enterprise. Open source* yang dikembangkan memanfaatkan teknologi *Software Defined Networking* (SDN) yang memisahkan fungsi kontrol dan pengiriman data dalam pengiriman paket dalam jaringan sehingga pengguna memiliki kontrol atas lalu lintas jaringannya. Adapun *OpenFlow* sebagai protokol SDN paling umum digunakan kemudian digunakan untuk membangun *open network* tersebut. Teknologi Internet dan penggunaan perangkat *mobile* yang berkembang pesat secara bersamaan menyebabkan ketiga kebutuhan tersebut harus dikembangkan secara bersamaan, salah satunya dalam bentuk teknologi jaringan nirkabel berbasis *open mobile*. Penelitian ini akan fokus pada pengembangan jaringan *Wi-Fi* berbasis *OpenFlow* dengan evaluasi pada proses *handoff* secara horizontal menggunakan skema *fast handoff* mengingat belum diterapkannya skema ini dalam komunikasi multimedia secara *real-time.* Hasil pengujian yang dilakukan pada *testbed* sederhana memperlihatkan bahwa *delay* proses *handoff* pada jaringan *Wi-Fi* berbasis *OpenFlow* adalah sebesar 79,9 milidetik atau 21% lebih cepat dibanding *delay handoff* jaringan “tradisional”. Aliran data saat komunikasi terjadi juga lebih stabil akibat adanya *flow* yang diterapkan di tiap *switch* berbasis *OpenFlow.* Namun hasil pengujian harus diteliti lebih lanjut akibat kondisi *testbed* yang kurang stabil, *tools packet capturer* yang belum memenuhi standar, dan perumusan *flow* yang lebih baik.

Kata kunci: OpenFlow, *access point*, OpenDayLight, *delay,* *handoff*

ABSTRACT

|  |  |  |
| --- | --- | --- |
| Nama | : | Isyana Paramitha |
| Program Studi | : | Computer Engineering |
| Judul | : | Performance Analysis of Fast Handoff Scheme Mechanism on OpenFlow-based Wi-Fi Network |

Research in communication network has the limit due to its problem of the supply frequency and equipment. To overcome this problem, open source can be the solution to build a helpful test bed for the research or academic purpose. Open source network can be developed using Software Defined Network (SDN) which has been continuosly developed due enormous number of installed base equipment and protocols that are inflexible, predefined, and fixed since SDN offers a flexible, dynamic, and programmable functionality of network systems. By using OpenFlow as its protocol, we can program the network flow in a flow table on different switches and routers. This research approches an OpenFlow-based WiFi environment using OpenFlow-based Access Point (OFAP) and OpenFlow controller. Through this system we expect to achieve high performance and reliability in in real-time traffic (e.g: video streaming) over WLAN, by reducing the handoff delay compared to normal WiFi environment. Each OFAP is deployed at two different rooms and being performed by several experiments to evaluate handoff delay. The result of this experiment is the handoff delay between OFAPs is 24% faster than handoff delay between traditional AP with average of 79.9 miliseconds. OpenFlow-based network also more stable than because of installed flows given to each packets. The discovered value needs to be examined further due to unstable *testbed,* deficiency in packet capturer tools, and better formulation towards installed flows.

Keywords: OpenFlow, *access point*, OpenDayLight, *delay,* *handoff*

# DAFTAR ISI

[HALAMAN PERNYATAAN ORISINALITAS iii](#_Toc406494747)

[HALAMAN PENGESAHAN iv](#_Toc406494747)

[KATA PENGANTAR v](#_Toc406494747)

[HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI SKRIPSI vi](#_Toc406494747)

[ABSTRAK vii](#_Toc406494748)

[ABSTRACT viii](#_Toc406494748)

[DAFTAR ISI ix](#_Toc406494750)

[DAFTAR GAMBAR x](#_Toc406494751)

[DAFTAR TABEL x](#_Toc406494751)i

[BAB 1 PENDAHULUAN 1](#_Toc406494753)

[1.1 Latar Belakang 1](#_Toc406494755)

[1.2 Perumusan Masalah 2](#_Toc406494755)

[1.3 Tujuan Penelitian 2](#_Toc406494756)

[1.4 Batasan Masalah 3](#_Toc406494757)

[1.5 Sistematika Penulisan 3](#_Toc406494758)

[BAB 2 TINJAUAN PUSTAKA 4](#_Toc406494760)

[2.1 Konsep Jenis Jaringan 4](#_Toc406494763)

[2.2 Proses *Handoff* 5](#_Toc406494764)

[2.3 Protokol *Fast Handoff* 9](#_Toc406494765)

[2.4 *Software Defined Networking* 19](#_Toc406494766)

[2.5 *OpenFlow* 20](#_Toc406494767)

[BAB 3 PERANCANGAN 21](#_Toc406494770)

[3.1 Perancangan Topologi 21](#_Toc406494773)

[3.2 Perangkat Pendukung 23](#_Toc406494776)

[3.2.1 Perangkat Keras 23](#_Toc406494777)

[3.2.3 Perangkat Lunak 25](#_Toc406494779)

[3.3 Skenario Pengujian 27](#_Toc406494776)

[BAB 4 HASIL PENGUJIAN DAN ANALISIS 29](#_Toc406494780)

[4.1 Konfigurasi *Testbed* 29](#_Toc406494773)

[4.2 Pengambilan Data 33](#_Toc406494773)

[4.4 Analisis Data 37](#_Toc406494773)

[4.5 Analisis Proses *Handoff* 45](#_Toc406494773)

[4.6 Struktur *Test bed* 47](#_Toc406494773)

[BAB 5 KESIMPULAN 48](#_Toc406494780)

[5.1 Kesimpulan 48](#_Toc406494773)

[5.2 Saran 49](#_Toc406494773)

[DAFTAR PUSTAKA](#_Toc406494782) 49

[Lampiran](#_Toc406494782) 52

# DAFTAR GAMBAR

[Gambar 2.1 Aliran pesan handoff pada jaringan IEEE 802.11 9](#_Toc406495232)

[Gambar 2.2 Skema *NG-pruning* 11](#_Toc406495233)

[Gambar 2.3 Skema *channel mask* 13](#_Toc406495234)

[Gambar 2.4 Operasi *SyncScan* 14](#_Toc406495235)

[Gambar 2.5 Skema FHR 16](#_Toc406495235)

[Gambar 2.6 Skema PNC 17](#_Toc406495235)

[Gambar 2.7 Skema SNC 18](#_Toc406495235)

[Gambar 2.8 Arsitektur SDN 19](#_Toc406495235)

[Gambar 2.9 Aliran data *OpenFlow switch* 21](#_Toc406495235)

[Gambar 2.10 Proses penanganan paket masuk ke *OpenFlow switch* 21](#_Toc406495235)

[Gambar 3.1 Rancangan Topologi 22](#_Toc406495236)

[Gambar 3.2 Lapisan SDN pada OpenDayLight 24](#_Toc406495237)

[Gambar 3.3 Arsitektur *Quagga* 26](#_Toc406495238)

[Gambar 3.4 *Screenshot* aplikasi tPacketCapture saat memulai dan berproses 27](#_Toc406495239)

[Gambar 3.5 Alur pelaksanaan pengujian dan pengambilan data](#_Toc406495239) 28

[Gambar 4.1 Deteksi OpenDayLight terhadap OFAP1 dan OFAP2](#_Toc406495239) 30

[Gambar 4.2 Aliran komunikasi](#_Toc406495239) 31

[Gambar 4.3 Tabel flow](#_Toc406495239) 31

[Gambar 4.4 Screenshot jendela *Conversations*](#_Toc406495239) 36

[Gambar 4.5 Komunikasi *Ethernet*](#_Toc406495239) 37

[Gambar 4.6 Grafik hasil *delay* pada perangkat *mobile*](#_Toc406495239) 37

[Gambar 4.7 Grafik hasil *delay* pada perangkat *laptop*](#_Toc406495239) 38

[Gambar 4.8 Hasil *packet capture*](#_Toc406495239) 39

[Gambar 4.9 Kondisi lalu lintas jaringan saat jaringan terhubung](#_Toc406495239) 40

[Gambar 4.10 Grafik *jitter*](#_Toc406495239) 40

[Gambar 4.11 Grafik besar paket yang diterima](#_Toc406495239) 41

[Gambar 4.12 Grafik banyak paket yang diterima pada lalu lintas TCP](#_Toc406495239) 43

[Gambar 4.13 *Screenshot* paket *OpenFlow* dalam aliran paket kontroler](#_Toc406495239) 44

[Gambar 4.14 Proses *handoff*](#_Toc406495239) 45

[Gambar 4.15 Denah penempatan perangkat jaringan](#_Toc406495239) 47

# DAFTAR TABEL

Tabel 2.1 Perbandingan kualitas *wireless* dan *wired* 6

Tabel 4.1 Kronologi pengambilan data melalui perangkat *mobile* 33

Tabel 4.2 Hasil perhitungan ∆t paket pada perangkat *mobile* 38

Tabel 4.3 Hasil perhitungan ∆t paket pada perangkat *laptop* 39

Tabel 4.4 Protokol yang berjalan pada kontroler 42

Tabel 4.5 Hasil pengukuran *delay handoff* pada [33] 46

# PENDAHULUAN

Pada bab ini akan dijelaskan mengenai latar belakang dari penelitian ini, perumusan masalah, tujuan dari penelitian, batasan masalah, serta sistematika penulisan.

1. **Latar Belakang**

Perkembangan teknologi perangkat jaringan dan perangkat komunikasi pengguna, serta semakin mudahnya akses terhadap jaringan telah mengembangkan Internet, yang merupakan salah satu media yang berperan penting dalam bidang telekomunikasi, saat ini memegang peranan penting dalam kehidupan masyarakat. Secara spesifik, perangkat *mobile* merupakan jenis perangkat yang perkembangannya paling tajam. Terdapat 6.8 milyar penduduk dunia dimana 4 milyar diantaranya telah memiliki telepon seluler, bahkan jumlah telepon seluler saat ini adalah lima kali jumlah PC yang dimiliki oleh masyarakat saat ini[1]. Dari jumlah pengguna telepon seluler tersebut, 45 juta warga Amerika menggunakan telepon seluler berkategori *smart phone* untuk mengakses Internet[2].

Seiring dengan perkembangan teknologi pada perangkat pengguna, jaringan, khususnya jaringan Wi-Fi juga mengalami peningkatan kecepatan mulai dari 54 Mbps pada 802.11a, 450 Mbps pada 802.11n, hingga 1.3 Gbps pada 802.11ac di tahun 2012. Hingga 2017, diperkirakan 71% komunikasi menggunakan telepon seluler akan menggunakan Wi-Fi[3]. Namun satu jaringan Wi-Fi tidak dapat mencakupi area yang amat luas. Sehingga saat pengguna telepon seluler sedang terhubung pada satu jaringan Wi-Fi dan bergerak ke area lain sehingga menimbulkan degradasi kualitas sinyal, akan lakukan proses *handoff*. Dalam proses ini, pengguna dengan *smart phone* yang terhubung dengan suatu Access Point (AP) dalam suatu jaringan Wi-Fi akan berpindah ke AP lainnya dimana pengguna sementara tidak akan terhubung dengan Internet dan hanya melakukan pertukaran *frame* manajemen 802.11 untuk membentuk sesi komunikasi baru dengan AP yang baru. Namun untuk memperbaharui kualitas komunikasi tersebut, proses *handoff* harus melalui empat fase dimana setiap fasenya berfungsi untuk menjalankan pengenalan komunikasi dan proses *overhead* yang menimbulkan konsekuensi terjadinya *delay*.

Semakin berkembangnya penggunaan Wi-Fi di masyarakat, maka proses *handoff* sebagai *mobility support* menjadi salah satu isu terpenting dalam teknologi 802.11. Telah banyak penelitian dan pembuatan skema untuk meminimalisir *delay* yang terjadi dalam skema *fast-handoff* namun masing-masing memiliki kelebihan dan kekurangan, sehingga masih dibutuhkan perbaikan dalam penanggulangan *delay* tersebut.

Perkembangan teknologi jaringan juga mencapai titik pengenalan teknologi Software Defined Network (SDN), sebuah arsitektur jaringan yang dinamis, mudah untuk dikelola dengan kebutuhan biaya yang lebih efektif, dan mudah beradaptasi dengan kondisi lingkungan. SDN memisahkan fungsi pengontrol jaringan dengan fungsi pengiriman data sehingga menyebabkan jaringan dapat secara langsung diprogram sesuai dengan servis jaringan yang ada. Protokol yang digunakan dalam SDN adalah OpenFlow. Dengan banyaknya keuntungan dari teknologi ini, SDN tentu akan menjadi solusi untuk pengelolaan jaringan yang lebih baik. Penggunaannya akan terus meluas, mulai dari area penelitian hingga perusahaan *enterprise* dan pada akhirnya masyarakat luas secara keseluruhan.

Hingga saat ini pengaplikasian implementasi teknologi SDN telah dilakukan secara bertahap dalam teknologi komunikasi “tradisional”, termasuk salah satunya jaringan Wi-Fi. Melihat potensi dari pemanfaatan teknologi ini, bukan tidak mungkin jaringan Wi-Fi berbasis OpenFlow adalah solusi dari usaha peminimalisiran total waktu *delay* dalam proses *handoff.*

1. **Perumusan Masalah**

Masalah yang akan dibahas pada penulisan ini adalah :

1. Konsep dasar *Mobile Handoff*
2. Prinsip kinerja Software Defined Network (SDN) dengan protokol OpenFlow
3. Pengaplikasian OpenFlow dalam jaringan Wi-Fi
4. Evaluasi proses *mobile handoff* dengan skema *fast handoff* berdasarkan belum adanya protokol *fast handoff* yang tersedia dalam jaringan Wi-Fi untuk aplikasi multimedia *real time*
5. **Tujuan Penelitian**

Tujuan dari penelitian ini adalah :

1. Evaluasi implementasi SDN dalam jaringan Wi-Fi
2. Membangun jaringan *mobile* terbuka (*open mobile network*) dengan jaringan Wi-Fi terbuka menggunakan OpenFlow
3. Mengevaluasi proses *mobile handoff* dengan skema *fast handoff* antar dua *access point* berbasis OpenFlow
4. **Batasan Masalah**

Penelitian ini dilakukan dengan menggunakan *testbed*, perangkat *mobile* pengguna jaringan serta perangkat lunak untuk menjalankan fungsi kontrol jaringan serta fungsi analisis proses *handoff*. Topologi yang dibangun terdiri dari satu server sebagai pengontrol jaringan sebagai Access Controller (AC) dengan menggunakan perangkat lunak OpenDayLight, dua buah *wireless router* sebagai Access Point (AP), dan satu buah perangkat *mobile* sebagai pengukur. Pengguna akan terhubung dengan masing-masing AP. Evaluasi dilakukan pada proses *handoff* jaringan secara horizontal saat pengguna berpindah dari AP pertama ke AP keduadengan skema *fast handoff.*

1. **Sistematika Penulisan**

Laporan ini terbagi dalam 5 bab, dimana pada tiap-tiap bab akan menjelaskan hal-hal berikut:

BAB 1 PENDAHULUAN

Bab ini menjelaskan tentang latar belakang, tujuan penelitian, batasan masalah, dan sistematika penulisan.

BAB 2 TINJAUAN PUSTAKA

Bab ini akan menjelaskan tentang teori-teori yang berkaitan dengan masalah yang dibahas dan berperan sebagai dasar teori dalam penelitian ini.

BAB 3 PERANCANGAN

Bab ini menjelaskan tentang rancangan sistem yang akan dibuat serta hardware dan software yang akan digunakan untuk mendukung penelitian ini.

BAB 4 PENGUJIAN DAN ANALISIS

Bab ini berisi tentang hasil pengujian yang dilakukan beserta analisis yang didapatkan dari hasil pengujian tersebut.

BAB 5 PENUTUP

Bab ini berisi tentang kesimpulan dari pengujian yang dilakukan serta solusi yang dapat diberikan berdasarkan kesimpulan.

# TINJAUAN PUSTAKA

Pada bab ini akan dijelaskan materi-materi yang menjadi dasar dilakukannya penelitian ini yang meliputi konsep dasar dari jenis jaringan *wired* dan *wireless*, proses *handoff* dengan bermacam-macam jenis *delay,* penanggulangan *delay* pada proses *handoff,* dan SDN serta OpenFlow sebagai protokol SDN.

1. **Konsep Jenis Jaringan**

Secara umum jaringan komputer dapat berupa jaringan *wired,* dimana untuk membangun koneksi diperlukan suatu media untuk menghubungkan antar perangkat, dan jaringan *wireless,* yang memanfaatkan gelombang radio sebagai media transmisinya. Terminologi *wired* dapat juga sebagai batas antar jaringan *wireless.* Umumnya jaringan *wired* menggunakan kabel Ethernet untuk mengirim data antar PC yang terhubung. Pada jaringan *wired* yang kecil, satu router dapat digunakan untuk menghubungkan seluruh komputer. Jaringan yang lebih besar biasanya memanfaatkan beberapa *router* dan *switch* yang terhubung satu sama lainnya. Umumnya topologi tersebut menhubungkan salah satuperangkat jaringan dengan kabel modem, jalur T1, atau tipe koneksi Internet lainnya yang menyediakan akses Internet ke seluruh perangkat yang terhubung dalam jaringan. *Wired* juga dapat mengacu pada perangkat *input*, mengingat tren *wireless keyboard* dan *mouse* yang semakin meningkat.

Semakin banyaknya perangkat yang terhubung secara *wireless* tidak benar-benar menghilangkan penggunaan jaringan *wired* terutama pada bidang *enterprise* mengingat keunggulannya atas jaringan *wireless* yakni, potensi terjadinya interferensi sinyal yang jauh lebih kecil dibanding jaringan *wireless* dankecepatan jaringan *wired* yang relatif lebih besar dibanding *wireless* sehingga rasio transfer datanya akan lebih cepat. Faktor lebih dari jaringan ini adalah jaminan atas kontrol, *security*, dan *reliability;* serta membutuhkan biaya yang relatif lebih kecil dibanding jaringan *wireless.*

Wi-Fi, atau nama dari standar IEEE 802.11x, adalah teknologi jaringan *wireless* yang paling umum digunakan. Pada awalnya Wi-Fi hanya digunakan pada area berstandar 802.11b dengan frekuensi 2.4 GHz, namun saat ini Wi-Fi mencakup seluruh standar 802.11 mulai dari 11b, 11a, *dual-band,* dan sebagainya dengan tujuan untuk menghilangkan kebingungan dalam interoperabilitas WLAN[4]. Sesuai dengan definisi jaringan *wireless,* Wi-Fi bekerja tanpa menggunakan media transmisi fisik namun memanfaatkan teknologi frekuensi radio, suatu frekuensi dalam spectrum elektromagnetik yang berhubungan dengan propagasi gelombang radio. Saat arus frekuensi radio diberikan pada sebuah antena, medan elektromagnetik akan terbentuk dan dapat berpropagasi melalui ruang.

Fondasi dalam suatu jaringan Wi-Fi adalah Access Point (AP) yang fungsi utamanya untuk mem-*broadcast* sinyal *wireless* sehingga komputer dapat mendeteksi dan membangun koneksi. Namun komputer harus memiliki *network adapter* jika ingin terhubung pada jaringan Wi-Fi. Baik tidaknya Wi-Fi tergantung pada penyesuaian standar 802.11x yang digunakan dengan kebutuhan jaringan. Misalkan 802.11a yang memang memiliki kecepatan lebih namun harganya lebih mahal dibanding dengan 802.11b; sehingga 802.11a lebih banyak ditemui digunakan pada jaringan bisnis.

Tabel 2.1 Perbandingan kualitas *wired* dan *wireless*

|  |  |  |
| --- | --- | --- |
|  | ***Wired*** | ***Wireless*** |
| **Instalasi** | Kesulitan lebih tinggi dibanding *wireless* | Lebih mudah, namun harus berhati-hati pada interferensi sinyal |
| **Biaya** | Lebih sedikit | Lebih banyak |
| **Reliabilitas** | Tinggi | Cukup tinggi |
| **Performa** | Sangat baik | Baik |
| ***Security*** | Cukup baik | Cukup baik |
| **Mobilitas** | Terbatas | Sangat baik |

1. **Proses *Handoff***

Seperti yang dijelaskan pada [5] proses *handoff* pada jaringan IEEE 802.11 dapat dibagi menjadi dua tahap:

1. *Discovery*

Rasio signal-to-noise (SNR) dapat mengalami degradasi karena pergerakan (mobilitas) oleh perangkat *mobile* (MH)dan menyebabkan dimulainya prosedur *handoff.* Sebelum menutup koneksi pada Access Point (AP) yang terhubung saat ini, MH harus menemukan AP yang memiliki potensial untuk membangun koneksi. Hal ini dilakukan di lapisan *medium access control* (MAC) dengan menjalankan fungsi *scanning.*

Terdapat dua jenis *scanning* pada Wi-Fi: pasif dan aktif. *Scanning* pasif berarti MH akan memperhatikan sinyal untuk “mendengarkan” frame *beacon* (penanda). Frame ini menyediakan informasi *timing* dan keterangan kepada MH sebagai media pertimbangan AP manakah yang akan dipilih oleh MH untuk membangun koneksi. Standar 802.11 saat ini telah memiliki fitur *multiple channel.*

Secara spesifik, standar 802.11b dan 11g beroerasi pada pita dengan lebar 2.4 GHz dan menggunakan 11 kanal dari 14 yang tersedia. Sedangkan 802.11a beroperasi pada pita 5 GHz yang memiliki 32 kanal. Saat mode pasif sedang aktif, MH akan mendengarkan setiap kanal media fisik satu per satu dengan tujuan menemukan AP yang baru, sehingga mode ini menghasilkan *delay* yang cukup signifikan. Di sisi lain, mode aktif *scanning* melibatkan kegiatan transmisi frame *probe request* dari AP. Setelah seluruh kanal telah di-*scan,* MH akan mengumpulkan seluruh informasi dari AP yang tersedia yang kemudian akan dipilih oleh MH untuk membangun koneksi. Lebih detilnya, prosedur aktif *scanning* meliputi[6]:

1. Prosedur kanal normal, dimana prosedur CSMA/CA (carrier sense multiple access with collision avoidance) akan dijalankan untuk memperoleh kontrol media *wireless*
2. MH mentransmisikan frame *probe request* yang ditujukan ke alamat *broadcast-*nya
3. *Probe timer* dimulai
4. MH menunggu respon
5. Jika tidak ada respon yang diterima oleh *MinChannelTime*, kanal berikutnya akan di-*scan*
6. Jika respon *probe* yang diterima *MinChannelTime* lebih dari satu, maka MH akan menghentikan masuknya frame tersebut pada *MaxChannelTime* dan memroses semua informasi yang sudah diterima
7. Seluruh step tersebut akan diulangi untuk kanal berikutnya
8. Reassociation

Prosedur *reassociation*[[1]](#footnote-1) meliputi otentikasi dan re-otentikasi terhadap AP baru dan proses transfer informasi MH dari AP lama. Otentikasi adalah suatu proses dimana AP akan menerima atau menolak MH. MH akan memulai proses otentikasi dengan mengirimkan frame *authentication process* kepada AP yang berisi informasi identitas MH. Kemudian AP akan merespon dengan frame *authentication response* yang mengindikasikan informansi tolak atau terima. Saat otentikasi telah sukses, MH akan mengirimkan frame *reassociation request* yang kemudian dijawab dengan frame *reassociation response* yang berisi pengumuman penolakan atau penerimaan.

Keseluruhan *delay* pada *handoff* dapat dibagi menjadi tiga *delay: probe delay, authentication delay,* dan *reassociation delay.*

1. Probe Delay (Penundaan Pemeriksaan)

*Delay* ini tergantung pada mode *scan* mana yang digunakan (pasif atau aktif). Rata-rata *probe delay* saat mode pasif dapat direpresentasikan dalam sebuah fungsi atas interval frame *beacon* dan jumlah kanal yang tersedia. Lebih spesifik, jika interval *beacon* adalah 100 msec, maka rata-rata *probe delay* IEEE 802.11b dengan 11 kanal adalah 1100 msec dan 11a dengan 32 kanal sebesar 3200 msec. Dapat diingat bahwa *delay* untuk *switching* belum dimasukkan, sekitar 40-150 usec[7]. Sedangkap pada mode aktif, *probe delay* dapat ditentukan dari nilai *MinChannelTime* dan *MaxChannelTime* yang tergantung pada perangkat. Prosedur *scanning* aktif mengharuskan MH untuk me-*scan* seluruh kanal yang tersedia. Sehingga, *probe delay* (TA) dari *scanning* aktif dapat dituliskan:

Dimana N adalah jumlah kanal yang tersedia. Secara sederhana, metode untuk mengurangi *probe delay* ini adalah dengan mengurangi jumlah kanal yang harus diperiksa misalkan hanya dengan pemeriksaan atas kanal-kanal tertentu yang dipilih, bukan semuanya. Metode lain adalah dengan memperbaiki nilai *MinChannelTime* dan *MaxChannelTime* dengan tujuan untuk mengurangi waktu tunggu kanal.

1. Authentication Delay (Penundaan Otentikasi)

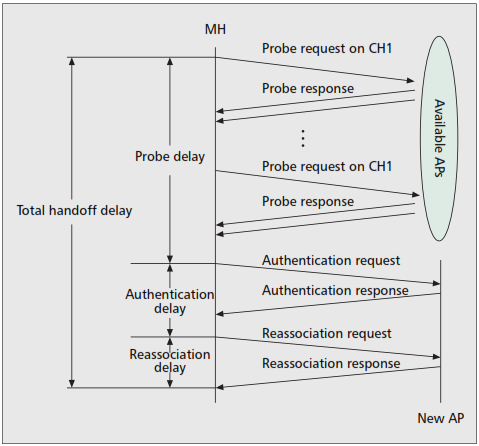
*Authentication delay* terjadi karena adanya pertukaran frame otentikasi. Secara umum, terdapat dua metode otentikasi: otentikasi *open-system* dimana AP akan selalu menerima MH tanpa memerlukan otentikasi apapun, atau dapat juga dengan melakukan filter pada alamat MAC (bukan merupakan bagian standar 802.11); dan otentikasi ­*shared-key*  berdasarkan *wired equivalent privacy* (WEP)[8], yang memerlukan AP dan MH untuk mengimplementasikan WEP. Otentikasi ini melakukan pertukaran empat jenis pesan:

1. Permohonan otentikasi MH terhadap AP dengan mengirimkan pesan *Challenge-Request*
2. AP mengirimkan nomor acak kepada MH melalui pesan *Challenge-Response*
3. MH akan menandai nomor acak tersebut menggunakan WEP, yang merupakan *secret-key* yang telah didistribusikan diantara kedua pihak melalui kanal yang aman sebelum digunakan. Kemudian mengirimkan pesan *Response* kembali ke AP
4. AP memverifikasi bahwa nomor acak telah ditandai dengan *key* yang benar dengan menghitung tanda tersebut yang kemudian dibandingkan dengan nilai yang diterima. Saat *key* telah berhasil diverifikasi oleh AP, AP akan mengotentikasi MH dengan mengirimkan pesan *Approval.*

Besar *authentication delay* sesuai dengan jumlah pesan yang dikirimkan antara AP dan MH. Sehingga, otentikasi *shared-key* menghasilkan *authentication delay* yang lebih panjang dibandingkan dengan otentikasi *open-system.* Lebih jauh, jika jaringan Wi-Fi memutakhirkan skema otentikasi seperti pada standar IEEE 802.11i (contohnya: IEEE 802.1x dan EAP-TLS[9]), diperlukan pesan yang lebih banyak. Contoh kasusnya adalah penggunaan sistem WLAN publik dengan skema otentikasi 802.1x (contoh: NeSpot di Korea[10]) mendorong adanya penelitian pada skema solusi untuk mengurangi *authentication delay.*

1. Reassociation Delay

*Reassociation* adalah proses pemindahan suatu asosiasi atau koneksi dari AP yang lama ke AP baru dalam satu *extended service set* (ESS). ESS adalah suatu kumpulan atas interkoneksi satu atau lebih *basic service set* (BSS) dimana suatu BSS adalah cakupan servis suatu AP. *Reassociation delay* muncul akibat adanya pertukaran frame *reassociation.* Mengikuti berhasilnya proses otentikasi, MH akan mengirimkan frame *reassociation request* kepada AP dan menerima frame *reassociation response* dan menyelesaikan proses *handoff.* Prosedur re-asosiasi hampir sama dengan prosedur asosiasi di jaringan pada umumnya. Namun di jaringan *backbone*, AP akan berinteraksi antar satu sama lain untuk mengirimkan frame yang berhubungan dengan re-asosiasi. Sehingga implementasi yang akan dating mungkin akan mencakup pesan tambahan protokol *inter-AP* (IAPP)[11] dalam fase re-asosiasi yang tentu menambah waktu *delay.*



Gambar 2.1 Aliran pesan *handoff* pada jaringan IEEE 802.11[11]

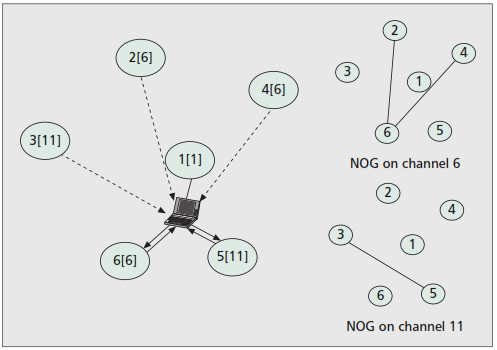
1. **Protokol *fast-handoff***

Tujuan utama protokol ini adalah untuk memastikan tiap pengguna (*user, client*) selalu terhubung dengan minimal satu *access point* setiap saat. Protokol ini terdiri dari banyak skema yang telah dikembangkan untuk meminimalisir *delay* pada proses *handoff* di jaringan IEEE 802.11. Dari banyaknya skema, penelitian ini memilih beberapa skema yang dianggap paling baik dan dibagi dalam dua kategori: skema pengurangan *probe delay* dan skema pengurangan *authentication/reassociation delay.* Bagian kedua menyatukan *authentication* dan *reassociation delay* karena operasinya hampir sama.

1. Mengurangi *probe delay*

Seperti yang diindikasikan pada [5], *probe delay* adalah kontributor utama *delay* dalam proses *handoff* sehingga usaha untuk menguranginya adalah krusial. Di [12], Velayos et al. mengajukan skema deteksi *fast-handoff* yang menggunakan distribusi *frame loss* (hilangnya frame) akibat *collision* (tabrakan) dengan tujuan untuk menentukan *handoff* optimal dengan pemicu waktu terhadap AP baru. Untuk mengurangi waktu deteksi *handoff,* MH memulai prosedur pengecekan pada kanal secepat mungkin setelah menganggap tabrakan data dapat tidak dimasukkan sebagai alasan kegagalan transmisi frame. Dengan kata lain, berdasarkan distribusi kemungkinan (probability distribution), jika re-transmisi suatu frame dan dua frame berikutnya gagal, MH menyimpulkan bahwa kegagalan fram disebabkan oleh pergerakan MH (bukan tabrakan) sehingga memerlukan proses *handoff* lebih jauh. Sebagai tambahan, [12] meningkatkan mode *scan* aktif dan menurunkan nilai baru untuk *MinChannelTime* dan *MaxChannelTime* dari hasil pengukuran dan model analitik. Secara spesifik, mereka menggunakan nilai yang lebih kecil yakni 1 ms dan 10.24 ms untuk *MinChannelTime* dan *MaxChannelTime*. Dengan menggunakan nilai *timer* ini, skema penyetelan (tuning) dapat mengurangi *probe delay.*

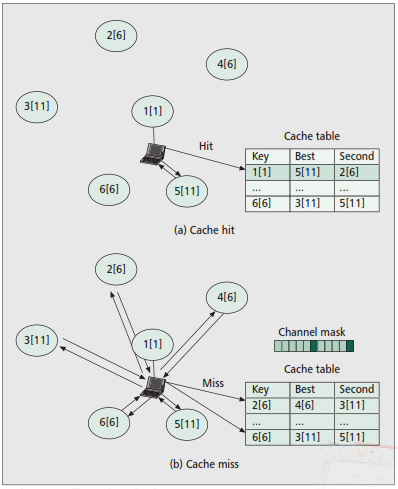
Referensi [13] mengajukan metode pencarian baru menggunakan *neighbor graph* (NG) dan *nonoverlap graph* (NOG). Skema ini, *NG-pruning,* fokus pada pengurangan jumlah kanal untuk diperiksa dan waktu tunggu di setiap kanal. Mereka mengajukan dua algoritma: *NG* dan *NG-pruning*. Pemikiran dasar dari algoritma tersebut adalah untuk memastikan butuh tidaknya sebuah kanal untuk diperiksa (menggunakan algoritma NG) dan apakah MH harus menunggu pesan *probe response* pada suatu kanal sebelum habisnya *MaxChannelTime* (menggunakan algoritma *NG-pruning*). NG menggambarkan hubungan *handoff* antar AP yang bersebalahan. Dengan menggunakan NG, kumpulan kanal yang sedang beroperasi pada AP yang bertetangga dapat dipelajari. Berdasarkan informasi ini, suatu MH dapat menentukan penting tidaknya suatu kanal untuk diperiksa. Di sisi lain, NOG menggambarkan hubungan *non-overlap* antara AP. Dua AP dianggap *non-overlap* jika dan hanya jika MH tidak dapat berkomunikasi dengan kedua AP tersebut secara simultan dengan kualitas koneksi yang dapat diterima. Contohnya, jika jarak antar AP*i* dan AP*j* jauh, suatu MH akan dapat berasosiasi dengan salah satunya. Dalam kasus ini, AP*i* dan AP*j* adalah *non-overlap.* Sehingga, jika MH telah menerima frame *probe response* dari AP*i* mengimplikasikan bahwa MH tidak dapat menerima frame *response* dari AP*j* sesuai prinsip *non-overlap*. Sesuai dengan visi NOG, MH dapat memangkas beberapa AP yang *non-overlap* dengan beberapa AP yang sedang terkoneksi dan telah memberi respon.



Gambar 2.2 Skema *NG-pruning*[13]

Terdapat angka yang berada di dalam [] dan tidak, keduanya merepresentasikan kanal yang digunakan oleh AP dan ID AP. Contohnya, hanya terdapat tiga kanal (contohnya: 1, 6, dan 11) yang digunakan dan AP aktif (AP1) memiliki lima AP tetangga (AP2 hingga AP6). Informasi tetangga dapat dipelajari dengan konstruksi NG. Dengan memanfaatkan informasi tersebut, MH mengetahui jumlah kanal yang harus dicek hanya sebanyak dua kanal (misalkan kanal 6 dan 11). Di sisi lain, individual NOG telah dikonstruksikan pada setiap kanal (misalkan satu NOG pada kanal 6 dan satu NOG pada kanal 11). Mulanya, misal MH sedang mengecek kanal 6. Saat MH menerima pesan *probe response* dari AP6, MH akan menentukan bahwa tidak perlu untuk menunggu pesan *probe response* tambahan di kanal 6. Hal ini karena AP6 *non-overlap* terhadap AP2 dan AP4 meskipun mereka menggunakan kanal 6 juga. Setelah mengecek kanal 6, MH akan mengirim pesan *probe request* ke kanal 11. Kemudian MH menerima pesan *probe response* dari AP5 dan mengehentikan pengecekannya pada kanal tersebut karena AP3 *non-overlap* dengan AP5.

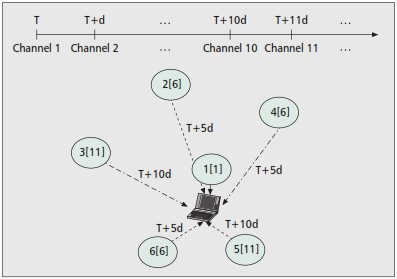
[6] mengajukan suatu algoritma *scanning* selektif dengan lima mekanisme *caching*, atau disebut sebagai skema *channel mask.* Skema ini hanya melakukan pengecekan pada sebagian kanal yang terpilih. Pemilihan kanal dilakukan dengan pertolongan *channel mask* yang telah ada saat *driver* dimuat (load) pada MH. Secara spesifik, *full-scan* dilakukan pertama kali dan *channel mask* kemudian dibangun menggunakan informasi yang didapat oleh ­*full-scan* tersebut. Pada IEEE 802.11b, hanya tiga kanal yang tidak tumpang-tindih (overlap) diantara sebelas kanal. Sehingga, dalam konfigurasi jaringan *wireless* yang baik, seluruh atau sebagian besar AP beroperasi pada kanal 1, 6, dan 11 yang menyebabkan *channel mask* dibentuk dengan mengombinasikan tiga kanal tersebut dan *full-scan* pertama kali dilakukan terhadap kanal-kanal tersebut. Dengan menggunakan *channel mask,* MH dapat mengurangi waktu yang tidak diperlukan untuk mengecek kanal yang tidak ada diantara AP yang bertetangga. Lebih jauh, untuk mengurangi *delay* pada *handoff,* mekanisme *cache* juga diperkenalkan. Ide dasar dari mekanisme ini adalah supaya masing-masing MH akan mencatat *handoff* yang telah dilakukan. Saat MH terhubung pada suatu AP, AP akan dimasukkan ke *cache* milik MH. Saat proses *handoff* diperlukan, MH pertama-tama akan mengecek apakah ada catatan yang berisi alamat MAC AP di *cache* tersebut. Jika ada, maka MH akan berasosiasi dengan AP tanpa perlu adanya prosedur pengecekan lebih jauh.



Gambar 2.3 Skema *channel mask*[6]

Gambar diatas mengilustrasikan operasi skema *channel mask.* Setiap MH memiliki *channel mask* yang dibentuk saat fase setup jaringan. Di waktu yang sama, setiap MH memiliki tabel *cache-*nya sendiri yang dibangun dan diperbaharui secara dinamis setiap terjadi proses *handoff.* Saat suatu MH berasosiasi dengan suatu AP, ID AP (misal alamat MAC) akan disimpan di *cache* sebagai kunci. Lebih jauh, dua AP dengan *received signal strength* (RSS) terbaik juga akan dicatat. Saat proses *handoff* diperlukan, pertama-tama MH akan mengecek data di tabel *cache* menggunakan alamat MAC AP yang sedang aktif sebagai pencari kunci. Jika ada catatan pada *cache* yang berhubungan dengan AP tersebut (misal *cache hit*), MH akan mencoba terhubung dengan AP bernilai RSS tertinggi. Jika asosiasinya berhasil, *handoff* berarti selesai dan latensi pada *handoff* dapat dikurangi secara signifikan. Hanya saat asosiasi pertama dan kedua gagal, MH akan melakukan pengecekan pada kanal terpilih menggunakan *channel mask*. Pada gambar 2.3 hanya tiga channel yang digunakan, sehingga akan mencukupi untuk MH melakukan pengecekan kanal walaupun jika terjadi luput pada pencatatan *cache.*

Pada [13], suatu skema *handoff* baru telah diajukan untuk mengurangi *probe delay.* Tidak seperti pada prosedur pengecekan yang terdefinisi di IEEE 802.11, SyncScan membolehkan MH untuk mengawasi jarak terhadap AP terdekat secara terus menerus. Dengan kata lain, MH akan beralih dari satu kanal ke kanal lainnya secara teratur untuk mencatat kekuatan sinyal dari kanal tersebut. Dengan melakukan hal tersebut, MH akan dapat memperbaharui informasi pada seluruh AP yang bertetangga. Lebih jauh, melalui pengawasan secara terus menerus terhadap beberapa AP, keputusan *handoff* akan menjadi lebih baik dan *authentication/reassociation delay* juga dapat dikurangi. Untuk meminimalisir hilangnya paket saat pengawasan periodik, *power saving mode* (PSM) pada spesifikasi IEEE 802.11 juga digunakan. Karena SyncScan berdasar pada pengawasan teratur terhadap beberapa AP, sinkronisasi waktu merupakan isu yang penting. Untuk sinkronisasi terhadap AP, *network time protocol* (NTP) dapat ditingkatkan. Di sisi lain, juka beberapa AP menggunakan kanal yang sama dan mengeluarkan *beacon* di saat yang bersamaan, teknik pengacakan akan digunakan.



Gambar 2.4 Operasi *SyncScan*[13]

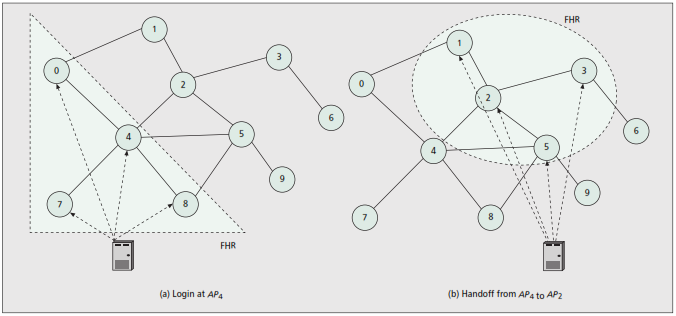
Gambar diatas menunjukkan diagram pengaturan waktu pada SyncScan, dimana *d* adalah parameter tidak tetap yang menentukan waktu *broadcast beacon.* Sebagai contoh, beberapa AP beroperasi pada kanal 1 mengirimkan *broadcast* frame *beacon* pada waktu *T,* saat beberapa AP pada kanal 2 akan melakukan hal yang sama pada T+*d,* AP pada kanal 3 pada T+2*d*, dan seterusnya. Dengan penjadwalan ini, MH akan berganti dari kanal yang sedang aktif (*c*) ke channel *c*+1 dan menerima frame *beacon* dari AP pada kanal tersebut. Operasi ini diulang terus menerus; sehingga MH dapat mempelajari informasi pada AP yang berdekatan. Karenanya, SyncScan membuat MH untuk mampu menentukan waktu kapan *handoff* harus dilakukan sehingga mengurangi *delay.*

Brik et al. memperkenalkan skema *handoff* menggunakan beberapa radio yang disebut *MultiScan*[14]. Mirip dengan SyncScan, MultiScan memperoleh informasi AP bertetangga dengan pemilihan *scanning.* Namun, MultiScan membutuhkan *interface* radio tambahan untuk melakukan *scanning.* Pada MultiScan, *interface* utama diasosiasikan dengan AP yang aktif dan digunakan untuk transmisi data. Di waktu yang bersamaan, *interface* kedua melakukan *scanning* pada kanal. Jika dibutuhkan proses *handoff* kepada AP baru, *interface* kedua akan terhubung ke AP baru selama *interface* utamanya masih melakukan data transmisi. Setelah asosiasi di *interface* kedua komplit, *interface* akan berpindah dari *interface* kedua ke utama. Hasilnya, *interface* utama baru tersebut akan melakukan transmisi data dan *interface* utama yang lama digunakan untuk *scanning* kanal. Disini MultiScan mendapat proses *handoff* yang memulai koneksi sebelum memutuskannya dengan memanfaatkan beberapa *interface* radio.

1. Mengurangi *authentication/reassociation delay*

Meskipun *probe delay* mengambil porsi besar dati total *delay* pada *handoff*, *authentication/reassociation delay* tetap harus diminimalisir untuk mencapai servis *mobile* yang *seamless* (lancar). Pada servis publik WLAN, skema otentikasi berbasis pada *server* otentikasi terpusat dengan tujuan servis yang aman dan akunting (*accounting*) yangefisien. Pada kondisi yang demikian, *authentication/reassociation delay* dapat lebih tinggi dibanding kondisi dimana prosedur otentikasi terbuka digunakan[15]. Dari tinjauan yang dilakukan terhadap metode otentikasi cepat pada jaringan IEEE 802.11 terdapat pada [16] dengan analisa berdasarkan pada arsitektur jaringan dan susunan *trust model*-nya. Dalam artikel ini akan didiskusikan beberapa skema berfokus pada model komunikasi.

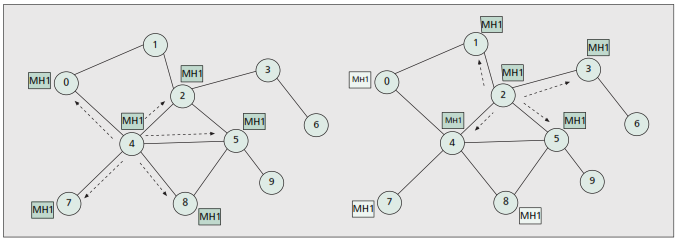
Pack et al. mengajukan skema *handoff* prediktif untuk mengurangi *authentication/reassociation delay* yang disebut sebagai skema FHR dalam [17]. Dalam skema ini, suatu informasi otentikasi MH didistribusikan secara proaktif ke beberapa AP sesuai dengan pola pergerakan MH dan kelas servisnya. Untuk memperkirakan pola tersebut, sebuah konsep *frequent handoff region* (FHR) kemudian diperkenalkan. FHR merupakan kumpulan AP yang memiliki kemungkinan tinggi untuk membangun koneksi dengan MH. FHR dikonstruksikan berdasarkan frekuensi *handoff* dan prioritas MH dalam sistem terpusat. FHR dapat dengan mudah diimplementasikan berdasarkan model IEEE 802.1x[18]. Dari beberapa studi pengukuran[19], jumlah AP yang terhubung dengan sebuah MH selama waktu servisnya biasanya sebanyak dua atau tiga AP. Sehingga dalam skema FHR, informasi otentikasi MH akan diberikan ke sebagian dari AP yang berdekatan yang berada di maksimum berjarak dua hop dari AP aktif.



Gambar 2.5 Skema FHR[17]

Gambar diatas menunjukkan operasi skema FHR. Dalam contoh tersebut, sebuah MH terhubung dengan AP dalam fase *login* dan membentuk sebuah FHR yang terdiri dari AP0, AP4, AP7, dan AP8. Sehingga informasi otentikasi akan didistribusikan kepada AP-AP tersebut terlebih dahulu. Jika MH berpindah ke salah satu AP tersebut, tidak perlu ada prosedur otentikasi kepada *server* otentikasi. Namun, jika MH berpindah ke AP lain (misal AP2 di gambar 2.5(b)) yang tidak termasuk dalam operasi otentikasi prediktif, otentikasi baru dan prosedur re-asosiasi harus dilakukan. Sebagai tambahannya, setelah melakukan proses *handoff* ke AP, sebuah FHR baru yang terdiri dari AP1, AP2, AP3, dan AP5 harus disusun.

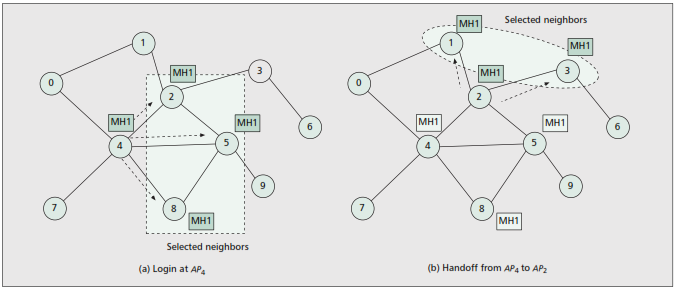
Alih-alih menggunakan sistem terpusat, sebuah skema proaktif berdasarkan struktur *cache* terdistribusi diperkenalkan dalam [20]. Skema ini dinamai *proactive neighbor caching* (PNC). PNC menggunakan grafik *neighbor* yang secara dinamis mencatat topologi pergerakan jaringan *wireless* untuk memosisikan konteks MH. Skema ini memastikan bahwa konteks MH akan selalu dikirimkan satu hop lebih maju, sehingga *delay* dapat dikurangi. Konteks yang dimaksud adalah kumpulan informasi mengenai sesi MH, *quality of service* (QoS), dan *security* [21]. Grafik *neighbor* dibangun menggunakan informasi yang saling ditukar saat *handoff* MH yang kemudian didistribusikan dan disimpan di setiap AP. Konteks AP yang disebarkan disimpan dalam *cache.* Konteks tersebut dapat digantikan dengan *cache replacement policy* (misalkan skema *least recently used* (LRU)) jika tidak ada kapasitas yang tersisa dalam *cache.* Baru-baru ini skema PNC telah dimasukkan dalam spesifikasi IAPP [11], sebagai protokol standar komunikasi antar AP.



Gambar 2.6 Skema PNC[20]

Gambar operasi PNC tersebut mengilustrasikan saat sebuah MH terhubung kepada AP4, AP ini akan menyebarkan konteks MH ke seluruh AP yang bertetangga (misalkan AP4, AP2, AP5, dan AP7). Saat MH berpindah ke AP2, tidak diperlukan prosedur otentikasi lebih jauh karena AP2 telah menerima konteks MH. Di waktu yang sama, konteks MH dipindah dari AP yang tidak berdekatan (AP0, AP7, dan AP8). Dalam skema PNC, suatu konteks MH disebarkan ke seluruh AP yang berdekatan saat proses (re)asosiasi baru dilakukan. Hal ini menyebabkan terjadinya tingginya *overhead* persinyalan (*signaling*), terutama saat terdapat banyak MH. Lebih jauh, studi pengukuran pada [19, 22] mengindikasikan bahwa meskipun dalam suatu kondisi dimana sejumlah AP dipasang, maksimum dua atau tiga AP adalah target utama dalam *handoff.* Maka dari itu, menyebarkan konteks MG kepada sebagian AP yang berdekatan merupakan langkah yang memadai untuk menyediakan servis *seamless* pada perangkat *mobile.*

Untuk mengurangi *overhead* pada *signaling* akibat transfer konteks, sebuah skema *neighbor caching* yang dikembangkan diperkenalkan sebagai *selective neighbor caching* (SNC) dalam [23]. Skema ini mengembagkan skema PNC dengan menambahkan konsep baru berupa *neighbor weight.* Konsep ini merupakan representasi probabilitas *handoff* pada setiap AP yang berdekatan. Berdasarkan *neighbor weight*, konteks MH disebarkan kepada beberapa AP berdekatan yang terpilih (misalkan AP-AP yang memiliki *neighbor weight* yang sama atau lebih tinggi dari batasan (*threshold*) yang ditentukan sebelumnya). Grafik *neighbor* dapat dengan mudah dibangun dengan mengawasi pola *handoff* antar AP.



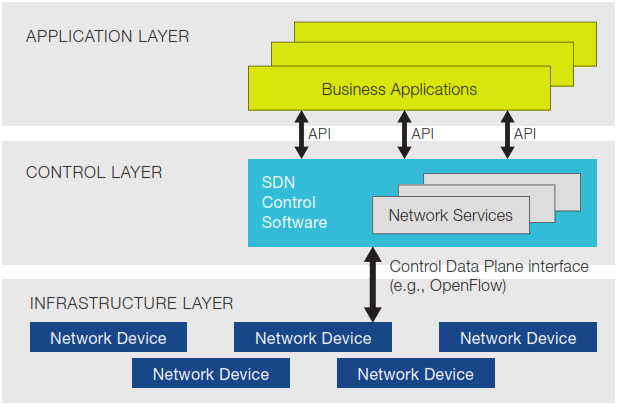
Gambar 2.7 Skema SNC[23]

Saat membandingkan gambar 2.7 dengan gambar 2.6, SNC menunjukkan proses transfer konteks yang lebih sedikit dibanding dengan PNC. Contohnya saat MH terhubung dengan AP4, hanya tiga *neighbor* (AP2, AP5, dan AP8) yang menerima konteks MH. Jika MH melakukan proses *handoff* terhadap salah satu *neighbor* tersebut, skema SNC menunjukkan pengurangan *delay* yang sama dengan skema SNC dimana skema SNC membutuhkan *authentication/reassociation delay* jika MH berpindah ke AP *non-neighbor.* SNC memiliki performa *handoff* yang hampir sama jika nilai batasannya ditentukan dengan hati-hati. Jika *cache* pada AP terbatas, skema SNC merupakan pilihan yang lebih baik dibanding skema PNC.

1. **Software Defined Networking (SDN)[24]**

Arsitektur jaringan saat ini, atau dapat disebut sebagai arsitektur jaringan tradisional, sudah dianggap tidak mampu untuk memenuhi kebutuhan *enterprise, carrier,* dan pengguna. Maka dari itu dibangunlah sebuah arsitektur jaringan baru sebagai bentuk transformasi jaringan tradisional yang disebut SDN. Dalam arsitektur SDN, kontrol dan bidang data dipisahkan serta “otak” dan keadaan (*state*) jaringan terpusat secara logis (*logical*). Hasilnya, *enterprise* dan *carrier* mendapatkan fitur untuk memprogram jaringan secara langsung yang belum pernah ada sebelumnya, fitur automasi, kontrol jaringan, serta skalabilitas dan fleksibilitas jaringan yang tinggi.

Migrasi kontrol jaringan, yang sebelumnya terikat dalam perangkat jaringan menjadi perangkat komputasi yang mudah diakses, menyebabkan dasar infrastruktur dibentuk untuk memenuhi kebutuhan aplikasi dan servis jaringan.



Gambar 2.8 Arsitektur SDN

Gambar diatas merupakan ilustrasi SDN dari sudut pandang secara logis (*logical*). “Otak” jaringan (secara logis) berada dipusat pengontrol SDN, yang menjaga pengawasan global terhadap jaringan. Hasilnya, jaringan akan terlihat sebagai satu *logical* switch dari sisi aplikasi dan perangkat *policy.* Maka dari itu, e*nterprise* dan *carrier* mendapatkan kontrol seluruh jaringan yang tidak lagi bergantung kepada vendor dari satu titik (*logical*), yang sangat menyederhanakan desain dan operasi jaringan. SDN juga menyederhanakan perangkat jaringan yang dibutuhkan, mengingat tidak dibutuhkannya lagi pemahaman dan proses ribuan implementasi protokol; hanya berupa instruksi yang diberikan oleh kontroler SDN.

SDN mentransformasikan prinsip pengiriman paket *end-to-end* tradisional: Paket dikirimkan melalui jaringan, dari pengirim ke penerima, tanpa ada modifikasi dengan pola lalu lintas jaringan yang melewati *middlebox*[25]*.* *Middlebox* ini berupa perangkat jaringan yang dapat mengubah, memfilter, menginspeksi, dan proses manipulasi lalu lintas lainnya dengan beragam tujuan selain mengirimkan paket data. Contoh dari perangkat ini misalkan translator alamat jaringan, *gateway* dengan aplikasi spesifikasi, *firewall,* IDS/IPS, dan sebagainya. Perangkat tersebut disebut sebagai *network ossification* (pengerasan jaringan), yang berarti instalasinya akan sulit untuk dimodifikasi dan membutuhkan biaya besar dalam pemeliharaan. Dengan transformasi tersebut, SDN membawa banyak keuntungan: Kontrol terpusat dan desentralisasinya menggunakan perangkat dengan jenis vendor yang berbeda, bidang data diatur dengan lapisan abstraksi API umum untuk seluruh perangkat SDN, serta menyederhanakan konfigurasi jaringan dan operasi dengan menerjemahkan konfigurasi otomasi tingkat tinggi menjadi pola pengiriman data yang spesifik pada elemen jaringan. Implementasi protokol dan servis jaringan juga lebih mudah pada SDN berkat abstraksi operasi yang tinggi.

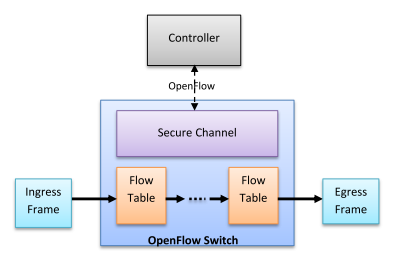
Namun disisi lain SDN juga memiliki kekurangan dalam tambahan *overhead* akibat peningkatan fleksibilitas dan fungsionalitas, sehingga terdapat penurunan performa dalam kecepatan proses dan *throughput.* Namun bukan berarti performa jaringan akan menurun secara keseluruhan, karena pada dasarnya SDN mengeksekusi servis dan pekerjaan jaringan dengan lebih sederhana.

1. **OpenFlow**

OpenFlow didesain sebagai paradigma jaringan baru yang memberi kesempatan bagi para peneliti untuk mengetes ide-ide baru dibawah kondisi yang realistik

OpenFlow[26] merupakan standar SDN yang terdiri dari dua entity utama: OpenFlow *switch* sebagai implementasi bidang data dan Kontroler sebagai bidang kontrol. Keduanya berkomunikasi melalui kanal terproteksi yang diimplementasikan dalam protokol special bernama OpenFlow.

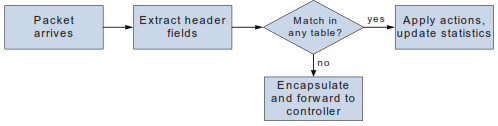
Setiap paket yang diterima di OpenFlow *switch* melewati satu set tabel aliran data (*flow table*), dimana satu atau lebih unit paket digunakan sebagai kata kunci pencarian. Saat unit tersebut cocok, maka paket tersebut akan diproses sesuai dengan servisnya atau akan diteruskan menuju tabel aliran data seterusnya yang pada akhirnya dikirimkan ke *port* keluar (*egress*), atau didrop.



Gambar 2.9 Aliran data OpenFlow *switch*

Kontroler mengonfigurasi operasi OpenFlow *switch* dengan memperbaharui tabel aliran data. Satu tabel berisi satu aturan serta satu set tindakan dan instruksi. Pencarian dilakukan pada setiap *frame* yang masuk sesuai dengan parameter aturan, diikuti dengan tindakan saat ditemukan data yang tepat. Setiap tindakan mendefinisikan operasi mana yang akan dijalankan pada *frame* yang masuk, dan instruksi mendefinisikan bagaimana tindakan tersebut akan dijalankan (*apply now, apply later,* dsb.).

Dalam kasus jika ada *frame* tidak teridentifikasi masuk dalam *switch, frame* tersebut akan dienkapsulasi dan dikirim ke kontroler yang akan memroses *frame* tersebut menggunakan logika yang lebih kompleks. Melalui itu, akan muncul perbaharuan pada tabel aliran data sehingga saat ada *frame* serupa yang masuk tidak perlu dikirimkan lagi ke kontroler.

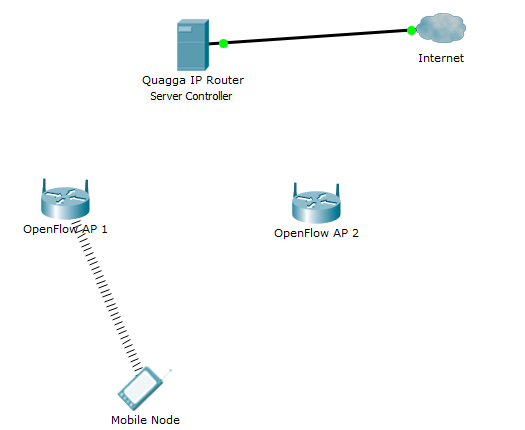


Gambar 2.10 Proses penanganan saat paket masuk ke OpenFlow *switch*

# PERANCANGAN

Pada bab ini akan dijelaskan mengenai perancangan topologi dari *testbed* yang akan dibangun, peralatan-peralatan yang mendukung, serta skenario pengujian yang akan dilakukan.

1. **Perancangan Topologi**



Gambar 3.1 Rancangan topologi

Jaringan yang digunakan pada pengujian ini secara garis besar merupakan jaringan dalam area dengan lingkup Wi-Fi berbasis OpenFlow. Topologi yang dibangun terdiri dari satu server, dua *wireless router,* dan satu perangkat *mobile.* Server akan bertindak sebagai *router* dengan instalasi Quagga sekaligus sebagai kontroler menggunakan OpenDayLight, serta penyedia servis *streaming* video untuk menguji koneksi *real-time.* *Wireless router* akan bertindak sebagai OpenFlow *Access Point* (OFAP).OFAP akan diletakkan di dua titik dan diset dengan posisi dimana dua sinyal tidak saling berinteferensi. Perangkat *mobile* dengan instalasi tPacketCapture akan merekam paket-paket di lalu lintas jaringan saat dihubungkan ke OFAP dan melalui proses *handoff.*

1. **Perangkat Pendukung**

Untuk memperoleh data pengukuran yang diinginkan sesuai dengan topologi yang digambarkan, akan digunakan beberapa peralatan pendukung, baik berupa *hardware* maupun *software* dengan spesifikasi seperti berikut.

1. Perangkat Keras

Daftar perangkat keras yang digunakan dalam pengujian ini adalah:

* 1. 1 buah Server

Server digunakan sebagai kontroler menggunakan perangkat lunak OpenDayLight serta *router* jaringan berbasis OpenFlow dengan instalasi perangkat lunak servis *routing* Quagga yang mendukung multi-protokol dengan zebra sebagai pengatur kernel *routing-*nya. Spesifikasi server yang digunakan adalah:

1. 4th Generation Intel® Core™ i5-4460 processor (6M Cache, up to 3.4 GHz)
2. 12GB Dual Channel DDR3 1600MHz (8GBx1 + 4GBx1)
3. 1TB 7200 rpm Hard Drive
   1. 2 buah *wireless router*

Perangkat ini digunakan untuk menjalankan fungsi OpenFlow *switch* yang berada di bawah *controller*. Untuk mendapatkan fungsi tersebut penelitian ini menggunakan OpenWRT dengan konfigurasi OpenFlow yang menjalankan fungsi *access point* berbasis OpenFlow. Spesifikasi *wireless router* yang digunakan adalah:

1. Buffalo WZR-HP-G450H
2. Wireless Standard IEEE802.11n, IEEE802.11g, IEEE802.11b
3. 3x3 TX/RX antennas
   1. 1 perangkat *mobile*

Perangkat ini dilengkapi dengan aplikasi *packet capturing* berbasis Android, tPacketCapture. Perangkat ini akan merekam paket yang melalui lalu lintas jaringan yang diperlukan khususnya saat proses *handoff* terjadi. Spesifikasinya adalah:

1. Android OS, v4.4.2 (KitKat), upgradable to v5.0 (Lollipop)
2. HSPA 42.2/5.76 Mbps, LTE Cat4 150/50 Mbps
3. Wi-Fi 802.11 a/b/g/n/ac, dual-band, Wi-Fi Direct, hotspot
4. Perangkat Lunak

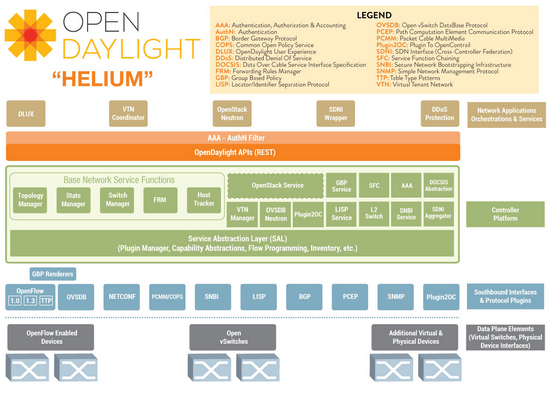
Daftar perangkat lunak yang digunakan dalam pengujian ini adalah:

1. Linux Ubuntu 12.04 LTS

Merupakan *Operating System* (OS) cuma-cuma yang dikembangkan oleh komunitas dan banyak dipakai untuk berbagai macam pengujian dikarenakan sifatnya yang *open source*. OS inilah yang digunakan pada server.

1. OpenDayLight

OpenDayLight adalah sebuah *open platform* untuk menjalankan fungsi pemrograman pada jaringan berskala dan berukuran apapun melalui aktifasi SDN.



Gambar 3.2 Lapisan SDN pada OpenDayLight

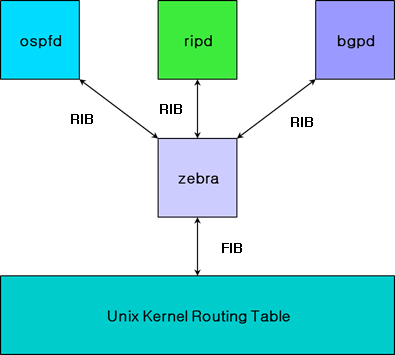
Lapisan atas terdiri dari aplikasi bisnis dan logika jaringan yang mengontrol dan mengawasi kegiatan jaringan, lapisan tengah merupakan *framework* dimana abstraksi SDN dijalankan dengan menyediakan satu set API-API umum di lapisan *interface* (umumnya disebut *northbound interface*) sambal mengimplementasikan satu atau lebih protokol untuk perintah dan kontrol perangkat keras fisik dalam jaringan (umumnya disebut *southbound interface*). Layer tengah inilah yang merupakan *platform* kontroler.

1. Pantou

Pantou akan mengubah suatu *wireless router/*AP menjadi OpenFlow *switch.* OpenFlow akan diimplementasi sebagai aplikasi diatas OpenWRT, sebuah sistem file bersama fungsi manajemennya sebagai *firmware* statis sehingga pengguna tidak lagi terbatas pada pilihan aplikasi dan konfigurasi milik vendor dan mengizinkan user untuk mengkustom beberapa paket dari perangkat *embedded* supaya cocok dengan aplikasi apapun. Dalam pembuatannya, OpenWRT harus terlebih dahulu diimplementasikan yang disesuaikan dengan jenis perangkatnya yang kemudian dikonfigurasi untuk mendapatkan fungsi OpenFlow.

1. Quagga

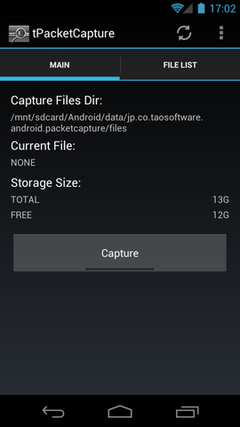
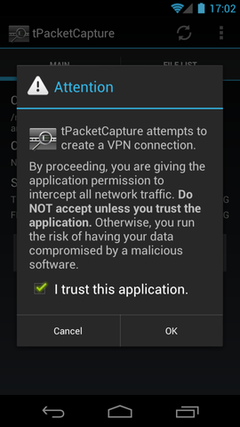
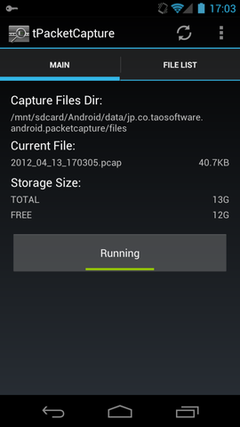
Quagga adalah suatu perangkat lunak *routing* yang menyediakan servis *routing* berbasis TCP/IP multi-protokol seperti RIPv1, RIPv2, RIPng, OSPFv2, OSPFv3, IS-IS, BGP-4, BGP-4+ hingga IPv6. Dengan *daemon* SNMP yang mendukung protokol SMUX dan AgentX, Quagga menyediakan protokol *routing* MIB. Arsitektur Quagga terdiri dari beberapa *daemon* yang bekerja sama dalam membentuk tabel *routing* dengan zebra sebagai kernel pengatur *routing.* Daemon yang tersedia antara lain ripd untuk protokol RIP, ospfd untuk protokol OSPF versi 2, bgpd, dan sebagainya. Zebra akan dibutuhkan saat adanya redistribusi rute antar protokol *routing* yang berbeda, sehingga akan lebih mudah untuk menambahkan prookol baru dalam suatu sistem *routing* tanpa memberi efek terhadap perangkat lunak lainnya. Pengguna dapat mengaktifkan *daemon* sesuai dengan kebutuhannya dan mengirim laporan *routing* langsung ke konsol *routing* utama.



Gambar 3.3 Arsitektur Quagga

1. tPacketCapture

Sebuah perangkat lunak untuk mencatat (*capture*) paket komunikasi pada perangkat tanpa harus di-*root* terlebih dahulu[30]. Aplikasi ini menggunakan servis VPN oleh Android. Data yang dicatat disimpan dalam file berformat .pcap yang kemudian dapat dianalisis oleh perangkat lunak *packet analyzer* seperti Wireshark. tPacketCapture tidak mengakses server eksternal apapun saat melakukan *packet capturing.* Aplikasi ini mencatat komunikasi TCP/UDP IPv4 sehingga paket ICMP seperti ping tidak akan dicatat. File .pcap disimpan di memori eksternal sehingga dapat langsung diambila oleh pengguna menggunakan USB ataupun langsung diunggah ke Dropbox atau di-*email.*

Gambar 3.4 *Screenshot* aplikasi tPacketCapture saat memulai dan berproses

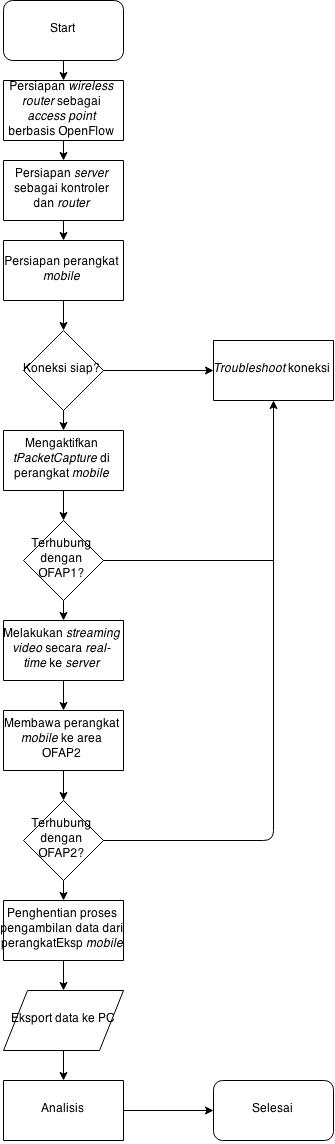
1. Wireshark

Wireshark adalah suatu aplikasi *packet analyzer* jaringan yang fungsinya adalah untuk mencatat paket-paket data dalam jaringan dan menampilkan detil paket tersebut. Perangkat ini dapat disebut sebagai alat pengukur yang digunakan untuk menguji hal apa yang terjadi dalam kabel jaringan, sama seperi voltmeter yang digunakan untuk menguji kabel listrik. Aplikasi ini merupakan aplikasi *open source* sehingga tidak memerlukan *cost* seperti halnya *analyzer* sebelum-sebelumnya[31].

Biasanya perangkat ini digunakan oleh administrator jaringan untuk *troubleshoot* masalah pada jaringan, teknisi *security* jaringan untuk menguji masalah *security, developer* untuk melakukan *debug* saat mengimplementasikan protokol, masyarakat yang ingin mempelajari internal protokol jaringan, dan sebagainya. Aplikasi ini dapat di-*install* di *platform* UNIX dan Windows, dapat mencatat paket data secara *real-time* pada sebuah *interface* jaringan, menampilkan paket data sekaligus dengan detil informasi protokol, pancatatan dapat difilter sesuai keinginan user, analisa data menggunakan statistik, dan sebagainya. Yang perlu diperhatikan adalah perangkat ini bukan merupakan *intrusion detection system* sehingga tidak dapat memberi peringatan saat seseorang tak dikenal mengakses jaringan, namun dapat memberitahu apa yang dilakukan oleh orang tersebut setelah beberapa saat. Wireshark tidak dapat memanipulasi jaringan seperti mengirim paket, fungsinya hanya sebatas “mengukur” jaringan.

1. **Skenario Pengujian**

Setiap AP berbasis OpenFlow (OFAP) akan diletakkan dalam dua ruangan yang berbeda, dengan jarak yang cukup jauh sehingga tidak ada interferensi sinyal diantara keduanya. Kedua AP tersebut dikontrol oleh server. Skema pengujian dan pengambilan data adalah sebagai berikut:



Y

N

Y

N

N

Y

Gambar 3.5 Alur Pelaksanaan Pengujian dan Pengambilan Data

Skenarionya adalah device *mobile* sebagai pengukur menggunakan aplikasi tPacketCaptureakan dihubungkan ke OFAP1 terlebih dahulu kemudian dibawa menjauh ke OFAP2 sambil melakukan *streaming* video kepada server. Saat OFAP berubah, proses *handoff* akan terjadi. Aplikasi akan memberi keluaran berupa file .pcapyang dapat dibuka melalui aplikasi Wireshark untuk kemudian dianalasis. Alamat IP yang digunakan server adalah 152.118.101.196. Sedangkan konfigurasi alamat pada kedua OFAP dilakukan secara statik untuk jaringan LAN dan dinamik untuk WAN. Konfigurasi lebih detil akan dijelaskan di bab selanjutnya.

# BAB 4

# HASIL PENGUJIAN DAN ANALISIS

Bab ini akan membahas hasil pengujian yang dilakukan menggunakan aplikasi tPacketCapture untuk merekam paket beserta keterangannya yang berjalan di lalu lintas jaringan yang kemudian dianalisis khususnya saat proses *handoff* antara dua OFAP terjadi sehingga waktu *delay* yang dibutuhkan akan didapatkan.

* 1. **Konfigurasi *Testbed***
     1. Quagga

Protokol *routing* yang diaktifkan dalam *testbed* ini adalah OSPF dan zebra sebagai kernel *routing suite.* Konfigurasi protokol tersebut dilakukan melalui parameter *yes* atau *no* di *file daemons.* Konfigurasi secara lengkap terlampir.

* + 1. OpenDayLight

*Tools* ini dapat dengan mudah digunakan dengan mengunduh dari halaman web resminya kemudian dieksekusi dengan perintah ./run.sh. Instalasi dan eksekusi terlampir.

* + 1. Pantou

Pantou merupakan konfigurasi OpenFlow 1.0 melalui distribusi OpenWrt. *Image* yang di-*flash* tergantung pada merk dan jenis *wireless router* dimana keterangan lengkapnya dapat dilihat di halaman web resmi OpenWrt. Setelah *image* berhasil di-*flash*, konfigurasi yang dilakukan terhadap kedua *access point* adalah:

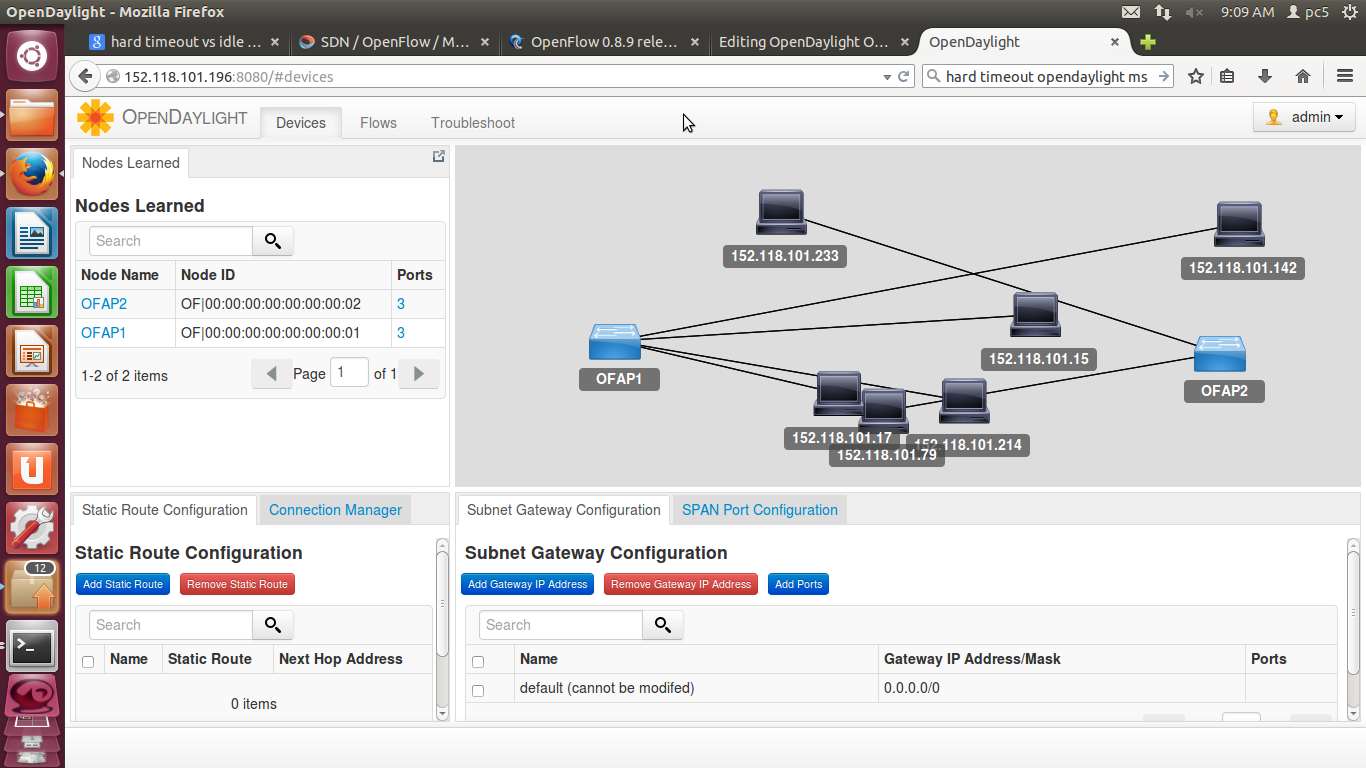
* Aktifasi *interface* dan *switch* melalui *file* konfigurasi *network. Interface* yang diaktifkan yaitu: loopback, LAN pada *interface* eth0.1, dan WAN pada eth0.2. LAN diberi alamat IP statis sedangkan WAN menggunakan DHCP. *Switch* dikonfigurasi dengan kondisi VLAN aktif, dengan dua VLAN yang masing-masing memiliki lima dan dua *port.*
* Konfigurasi *OpenFlow* juga dilakukan melalui *file* konfigurasi *openflow.* Contoh parameternya adalah *ofports* yang mengaktifkan eth0.1, eth0.2, dan wlan0 untuk koneksi *wireless;* serta parameter *ofctl* untuk membangun koneksi dengan perangkat kontroler dengan menyatakan alamat IP-nya. Baris perintahnya adalah 'tcp: 152.118.101.196:6633' dimana 152.118.101.196 adalah alamat IP kontroler dan 6633 menyatakan koneksi melalui *port OpenFlow.*

Langkah-langkah membangun *image* serta konfigurasi terlampir.

* + 1. Server

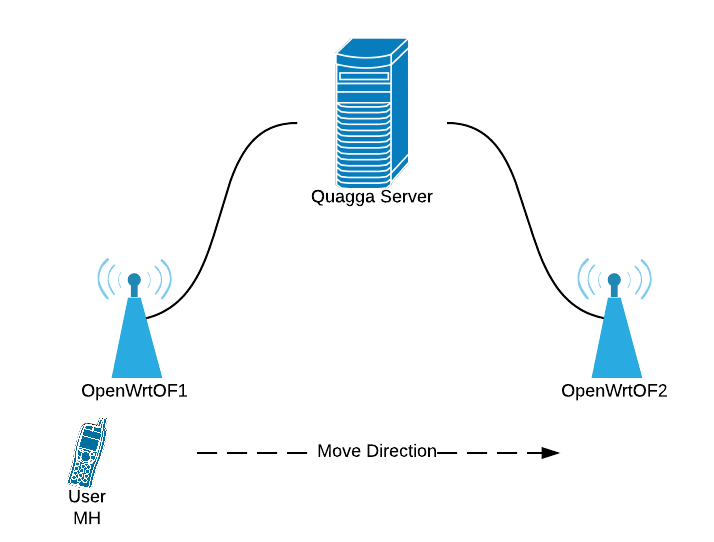
Server dipasang dengan alamat IP 152.118.101.196 dengan Quagga dan OpenDayLight yang diaktifkan. Server memanfaatkan VLC untuk menjalankan servis video *streamer* berbasis HTTP. Sehingga dapat diakses oleh perangkat *mobile* melalui http://152.118.101.196:1234 (proxy harus dinon-aktifkan). Port 1234 digunakan karena 8080 digunakan oleh OpenDayLight.

Konfigurasi aliran (*flow*) data sebagai *rules* dari jalannya paket diatur melalui OpenDayLight. *Tools* ini dapat diakses melalui aplikasi perambah *web* dengan mengetikkan alamat IP *loopback* dengan port 8080. Karena server ini diberi IP statik, maka untuk menjalankannya adalah melalui <http://152.118.101.196:8080> dengan eksekusi ./run.sh terlebih dahulu. Setelah pengguna login, jika *switch* OpenFlow memiliki konfigurasi yang benar (berada dalam satu segmen dengan kontroler dan memiliki konfigurasi alamat IP yang benar untuk kontroler) OpenDayLight akan secara otomatis mendeteksi *switch-switch* yang aktif.



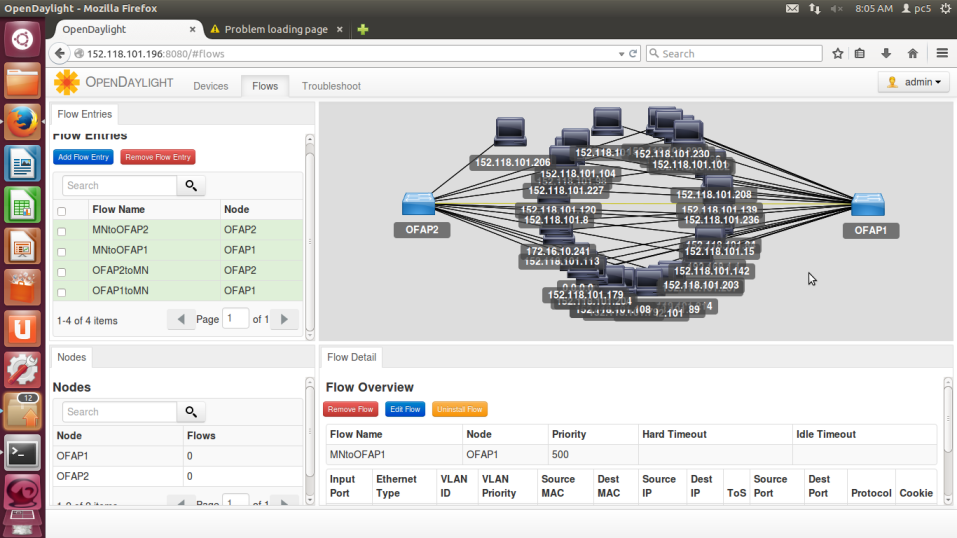
Gambar 4.1 Deteksi OpenDayLight terhadap OFAP1 dan OFAP2

Melalui *interface* inilah pengguna dapat mengimplementasikan aturan *flow.* Gambar dibawah menggambarkan aliran komunikasi yang ingin diimplementasikan:



Gambar 4.2 Aliran komunikasi

Sehingga penelitian ini akan mengonfigurasi *flow* pada masing-masing *switch* untuk “mengalirkan” data dari perangkat *mobile* ke masing-masing OFAP dan sebaliknya.Terdapat empat macam flow yang di-*install* melalui OpenDayLight.



Gambar 4.3 Tabel *flow*

*Flow* pertama adalah MNtoOFAP1, yakni aliran dari perangkat *mobile* (MN) ke OFAP1. *Node* yang dikonfigurasi adalah OFAP1 dengan *port input* eth0.1 karena MN akan terhubung dengan OFAP melalui *port* tersebut. *Actions* yang dipilih adalah *Add output port* dengan *port output* eth0.2 sebagai *port* WAN. *Flow* kedua adalah MNtoOFAP2 dengan konfigurasi yang sama dengan *flow* MNtoOFAP1. *Flow* ketiga dan keempat adalah aliran dari jaringan Internet ke MN. *Flow* OFAP1toMN mengonfigurasi *node* OFAP1 dengan *port input* eth0.2, dengan *action Add output port,* dan *port output* eth0.1. Pengaturan yang sama juga dikonfigurasikan di *flow* OFAP2toMN.

Seperti yang telah dibahas di bab II mengenai proses *handoff,* terdapat tiga macam delay: *probe, authentication,* dan *reassociation. Probe delay* tergantung nilai *MinimumChannelTime* dan *MaximumChannelTime* sehingga untuk meminimalisir *delay* harus ada perhitungan yang tepat untuk kedua *channel time* tersebut. Menurut [12] nilai optimal untuk *MinimumChannelTime* adalah 1 milisekon. Sedangkan untuk *MaximumChannelTime* adalah 10,24 milisekon. Parameter *MaximumChannelTime* dapat diatur melalui *OpenDayLight* melalui *prompt Hard Timeout* dan *Idle Timeout. Hard timeout* terjadi saat satu periode waktu telah habis, tidak peduli ada tidaknya paket yang cocok (terkirim/sampai) sedangkan *idle timeout* terjadi saat tidak ada paket yang cocok dalam satu periode waktu.

Karena waktu yang dianjurkan untuk *MaximumChannelTime* adalah 10,24 milisekon maka konfigurasi *idle timeout* harus dilakukan karena proses *handoff* terjadi saat paket tidak terkirim bukan karena dibatasi dalam satu periode waktu. Namun masalah yang belum dipecahkan adalah unit yang digunakan *OpenDayLight* berupa sekon. Saat coba memasukkan nilai 0,0... aplikasi tersebut tidak dapat mengeksekusinya.

* + 1. OpenFlow *Access Point*

Seperti yang disertakan diatas, konfigurasi kedua OFAP adalah identik kecuali pengalamatannya. Port OpenFlow yang diaktifkan adalah eth0.1, eth0.2, dan wan0. eth0.1 dan eth0.2 merupakan VLAN dari eth0. *Interface* eth0.1 diaktifkan untuk koneksi LAN dan *bridge* dengan alamat IP statik 192.168.1.1 untuk OFAP1 dan 192.168.2.1 untuk OFAP2. *Interface* eth0.2 digunakan sebagai koneksi WAN menggunakan alamat IP dhcp. VLAN diaktifkan pada *interface* switch0 untuk VLAN 1 dan VLAN 2. Terdapat 5 *port* yang diaktifkan di VLAN 1 dan 2 *port* di VLAN 2. Konfigurasi ‘ofctl’ yang menghubungkan ke PC *controller* kedua OFAPdiberi konfigurasi tcp:152.118.101.196 dengan port 6633, dengan alamat IP kontroler dan port openflow. Disediakan dua mode yakni *outofband* dan *inband*, namun OFAP dikonfigurasi menggunakan mode *outofband.*

Konfigurasi *no slicing* yang disertakan di bagian sebelumnya berarti menon-aktifkan fitur *slicing* oleh OpenFlow. Fitur ini dicoba pada OpenFlow 1.0 semata-mata sebagai langkah awal dari perwujudan *Quality of Service* (QOS) pada jaringan OpenFlow, yang jika diaplikasikan seperti halnya pada jaringan umum, akan membuat OpenFlow terlalu rumit sehingga menghilangkan tujuan awal dibuatnya *tools* ini. *Slicing* yang dimaksud disini berbeda dengan *network slicing* pada FlowVisor, meskipun keduanya berhubungan. *Slicing* adalah memberi garansi minimum *bandwidth* kepada tiap *queue* sedangkan FlowVisor “membagi” ruang aliran data menjadi beberapa bagian. Pada awalnya mekanisme ini dibuat untuk membuat satu ruang “isolasi” untuk beberapa eksperimen yang memiliki kebutuhan berbeda namun masih dilakukan pada satu eksperimen. Karena percobaan ini dibuat lebih sederhana dengan lalu lintas jaringan tidak banyak, maka mekanisme *slicing* tidak disertakan di konfigurasi switch OpenFlow.

* 1. **Pengambilan Data**

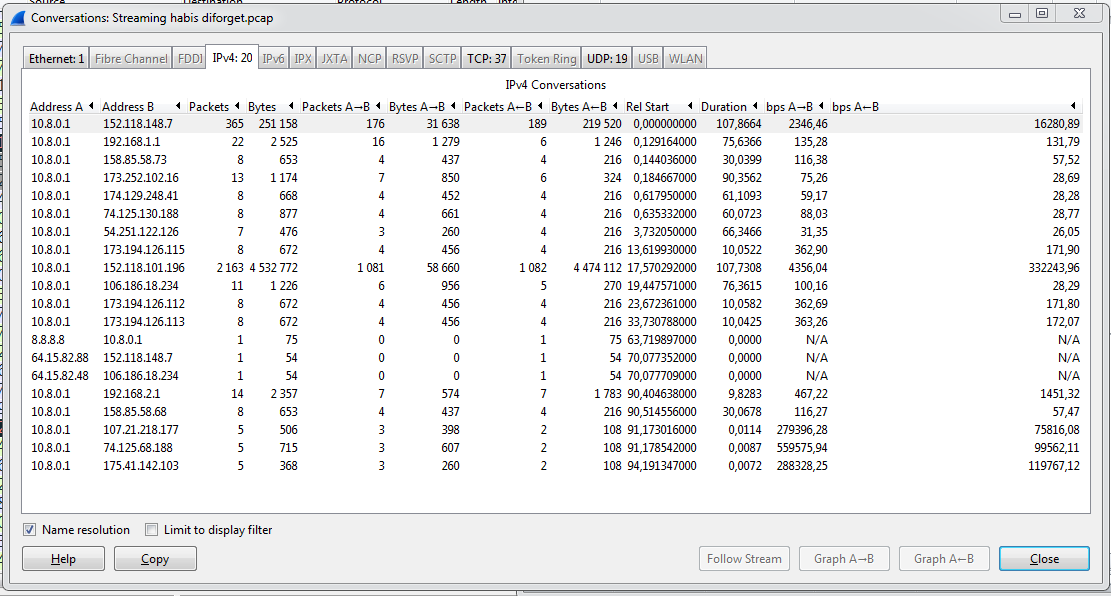
Iterasi pengambilan data yang dilakukan adalah sebanyak 28 kali. Perhitungan *delay* dilakukan berdasarkan jarak waktu paket TCP terakhir dari atau menuju ke perangkat *mobile* hingga paket pertama DNS yang dikirimkan menuju AP baru dari perangkat tersebut sebagai tanda bahwa perangkat telah terhubung ke AP baru.Rinciannya adalah sebagai berikut:

Tabel 4.1 Kronologi pengambilan data melalui perangkat *mobile*

|  |  |
| --- | --- |
| **Pengambilan ke-** | **Keterangan** |
| 1 | Hasil pengambilan perpindahan OFAP1 ke OFAP2 normal, dengan paket TCP dan DNS yang terekam dengan jelas |
| 2 | Hasil pengambilan perpindahan OFAP1 ke OFAP2 normal, dengan paket TCP dan DNS yang terekam dengan jelas |
| 3 | Hasil pengambilan perpindahan OFAP1 ke OFAP2 tidak terekam dengan baik, tidak ada paket yang menunjukkan bahwa perangkat berpindah dan terhubung dengan OFAP2 |
| 4 | Hasil pengambilan perpindahan OFAP1 ke OFAP2 tidak terekam dengan baik, tidak ada paket yang menunjukkan bahwa perangkat berpindah dan terhubung dengan OFAP2 |
| 5 | Proses pengambilan data tidak memenuhi parameter |
| 6 | Proses pengambilan data tidak memenuhi parameter |
| 7 | Proses pengambilan data tidak memenuhi parameter |
| 8 | Hasil pengambilan perpindahan OFAP1 ke OFAP2 normal, dengan paket TCP dan DNS yang terekam dengan jelas |
| 9 | Hasil pengambilan perpindahan OFAP1 ke OFAP2 tidak terekam dengan baik, tidak ada paket yang menunjukkan bahwa perangkat berpindah dan terhubung dengan OFAP2 |
| 10 | Hasil pengambilan perpindahan OFAP1 ke OFAP2 tidak terekam dengan baik, tidak ada paket yang menunjukkan bahwa perangkat berpindah dan terhubung dengan OFAP2 |
| 11 | Hasil pengambilan perpindahan OFAP2 ke OFAP1 normal, dengan paket TCP dan DNS yang terekam dengan jelas |
| 12 | Hasil pengambilan perpindahan OFAP1 ke OFAP2 tidak terekam dengan baik, tidak ada paket yang menunjukkan bahwa perangkat berpindah dan terhubung dengan OFAP2 |
| 13 | Hasil pengambilan perpindahan OFAP2 ke OFAP1 normal, dengan paket TCP dan DNS yang terekam dengan jelas |
| 14 | Hasil pengambilan perpindahan OFAP1 ke OFAP2 tidak terekam dengan baik, tidak ada paket yang menunjukkan bahwa perangkat berpindah dan terhubung dengan OFAP2 |
| 15 | Proses pengambilan data tidak memenuhi parameter |
| 16 | Proses pengambilan data tidak memenuhi parameter |
| 17 | Hasil pengambilan perpindahan OFAP1 ke OFAP2 tidak terekam dengan baik, tidak ada paket yang menunjukkan bahwa perangkat berpindah dan terhubung dengan OFAP2 |
| 18 | Hasil pengambilan perpindahan OFAP1 ke OFAP2 tidak terekam dengan baik, tidak ada paket yang menunjukkan bahwa perangkat berpindah dan terhubung dengan OFAP2 |
| 19 | Hasil pengambilan perpindahan OFAP1 ke OFAP2 tidak terekam dengan baik, tidak ada paket yang menunjukkan bahwa perangkat berpindah dan terhubung dengan OFAP2 |
| 20 | Hasil pengambilan perpindahan OFAP1 ke OFAP2 normal, dengan paket TCP dan DNS yang terekam dengan jelas |
| 21 | Hasil pengambilan perpindahan OFAP1 ke OFAP2 normal, dengan paket TCP dan DNS yang terekam dengan jelas. Namun paket TCP terakhir yang dikirimkan bukan berupa koneksi perangkat ke *server* |
| 22 | Hasil pengambilan perpindahan OFAP1 ke OFAP2 normal, dengan paket TCP dan DNS yang terekam dengan jelas |
| 23 | Hasil pengambilan perpindahan OFAP2 ke OFAP1 normal, dengan paket TCP dan DNS yang terekam dengan jelas |
| 24 | Hasil pengambilan perpindahan OFAP1 ke OFAP2 normal, dengan paket TCP dan DNS yang terekam dengan jelas. Namun paket TCP terakhir yang dikirimkan bukan berupa koneksi perangkat ke *server* |
| 25 | Hasil pengambilan perpindahan OFAP2 ke OFAP1 normal, dengan paket TCP dan DNS yang terekam dengan jelas |
| 26 | Hasil pengambilan perpindahan OFAP1 ke OFAP2 normal, dengan paket TCP dan DNS yang terekam dengan jelas |
| 27 | Hasil pengambilan perpindahan OFAP1 ke OFAP2 normal, dengan paket TCP dan DNS yang terekam dengan jelas |
| 28 | Hasil pengambilan perpindahan OFAP2 ke OFAP1 normal, dengan paket TCP dan DNS yang terekam dengan jelas |

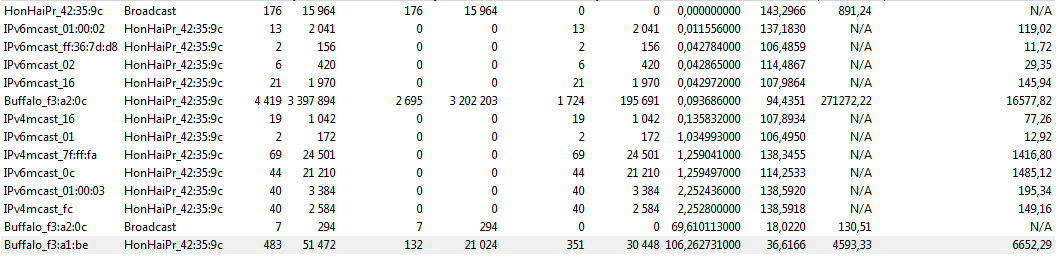
Dari tabel diatas dapat dilihat dari 28 hasil pengambilan data, lima data tidak dapat digunakan karena parameter yang tidak terpenuhi. Parameter yang tidak terpenuhi adalah tidak berjalannya *streaming* dengan baik serta tidak terjadinya proses *handoff* akibat ketidakstabilan *testbed.* Dari 23 hasil yang memenuhi parameter, terdapat sembilan data yang tidak terlihat perpindahan AP-nya secara jelas. Misalnya pada pengambilan data nomor 3, perpindahan dilakukan dari OFAP1 ke OFAP2 namun tidak ada paket DNS menuju ke 192.168.2.1 yang tercatat. Kemungkinan permasalahan ini terjadi terletak pada aplikasi *packet capturer* yang digunakan memiliki fungsi VPN. Sehingga saat perangkat yang terhubung dengan AP berbeda memiliki alamat IP yang berbeda, hal ini tidak dapat terlihat. Perhitungan *delay* pada sembilan data tersbut memiliki kemungkinan untuk dihitung dengan mengasumsikan letak proses *handoff,* namun untuk riset tentu hal ini tidak bisa dilakukan. Pada akhirnya hanya 14 data yang memenuhi parameter dari pengambilan data melalui perangkat *mobile* yang kemudian diolah menggunakan *Wireshark* untuk analisisnya. Pengambilan data tambahan juga dilakukan sebanyak empat kali menggunakan perangkat *laptop* mengingat keterbatasan aplikasi tPacketCapture yang tidak mampu menangkap *frame,* sedangkan proses *handoff* harus dievaluasi melalui *frame* manajemennya. Pengambilan data ini menggunakan *Wireshark.*

Empat belas data yang diambil melalui perangkat *mobile* memiliki pola yang kurang lebih sama. Misalkan pada pengambilan data 1 secara garis besar pencatatan dilakukan selama 2 menit 5 detik. Terdapat 2669 paket yang tercatat dimana hampir 99% terdiri paket TCP dan sisanya UDP untuk paket DNS. Paket TCP merupakan mayoritas karena lalu lintas yang dicatat adalah *streaming* video melalui Internet. Melalui fitur Statistics 🡪 Conversations pada Wireshark, lalu lintas yang terjadi antar dua perangkat *end point* secara spesifik dapat diamati:

Gambar 4.4 *Screenshot* jendela *Conversations*

Terdapat beberapa segmentasi paket: Ethernet, IPv4, TCP, dan UDP. Karena pengambilan data paket melalui satu *node* maka aliran Ethernet paket hanya terjadi dari 00:1a:11:00:00:01 ke 00:1a:11:00:00:02. Untuk segmen paket IPv4, sebagian paket berasal dari perangkat *mobile* ke server (152.118.101.196) dan ke Internet (152.118.148.7). Begitupun pada segmen TCP, port yang paling banyak dituju adalah *port* 1234 sebagai port akses *streaming* video. Koneksi perangkat *mobile* ke OFAP1 dan OFAP2 terlihat dalam pertukaran paket DNS di segmen UDP.

Seperti yang dijelaskan sebelumnya, pengukuran data dengan menggunakan media *laptop* dilakukan dengan pertimbangan *tools* untuk melakukat *packet capture* lebih baik dibanding aplikasi tPacketCapture. Metode yang digunakan sama persis dengan kedua pengukuran diatas namun paket pada *layer* 2 dapat direkam dengan baik. Pengambilan data dilakukan selama 2 menit 23 detik dengan total 5341 paket yang direkam. Komunikasi *layer* 3 sama dengan kedua paket sebelumnya. Sebagian besar komunikasi *layer* 2 yang terjadi adalah antara *laptop* dengan masing-masing AP, *laptop* untuk melakukan *broadcast,* dan masing-masing AP yang melakukan *broadcast.*



Gambar 4.5.Komunikasi Ethernet

* 1. **Analisis Data**

Menurut [32] suatu komunikasi VoIP, atau bisa diluaskan sebagai komunikasi multimedia *real-time* memerlukan waktu sebesar 100 milisekon (ms) sebelum koneksinya dianggap terputus. Sehingga data yang diharapkan bernilai <100 ms. Untuk data yang diambil menggunakan tPacketCapture analisis dilakukan berdasarkan perbedaan waktu (∆t) antara paket TCP terakhir yang dikirim atau diterima oleh perangkat *mobile* dengan paket DNS pertama yang dikirim ke AP baru di setiap hasil pengambilan data, kecuali data ke-21 dan 24 dimana koneksi TCP terakhir terjadi akibat aktifitas servis aplikasi komunikasi yang di-*install* di perangkat tersebut. Seperti yang telah dijelaskan sebelumnya, hanya ada 14 data yang menampilkan paket DNS secara jelas. Hasil pengambilan data adalah sebagai berikut:

Gambar 4.6.Grafik hasil *delay* pada perangkat *mobile*

Dapat dilihat bahwa data pada pengambile ke-21 dan 24 memiliki anomali. Hasil perhitungan *delay* yakni berupa rata-rata dari:

Tabel 4.2 Hasil perhitungan ∆t paket pada perangkat *mobile*

|  |  |
| --- | --- |
| **Pengambilan data ke-** | **Delay (ms)** |
| 1 | 90 |
| 2 | 61 |
| 8 | 118 |
| 11 | 84 |
| 13 | 31 |
| 20 | 110 |
| 21 | 4304 |
| 22 | 52 |
| 23 | 207 |
| 24 | 4181 |
| 25 | 50 |
| 26 | 65 |
| 27 | 23 |
| 28 | 68 |
| **Rata-rata** | **79,9** |

Data ke-21 dan 24 tidak dihitung mengingat kedua data tersebut memiliki anomali akibat berjalannya servis aplikasi yang terhubung ke Internet. Sedangkan pada paket yang di-*capture* oleh *laptop* menggunakan *Wireshark* sehingga terlihat komunikasi *frame-*nya adalah sebagai berikut:

Gambar 4.7.Grafik hasil *delay* pada perangkat *laptop*

Dengan hasil *delay* dari perhitungan rata-rata keempat data tersebut sebagai berikut:

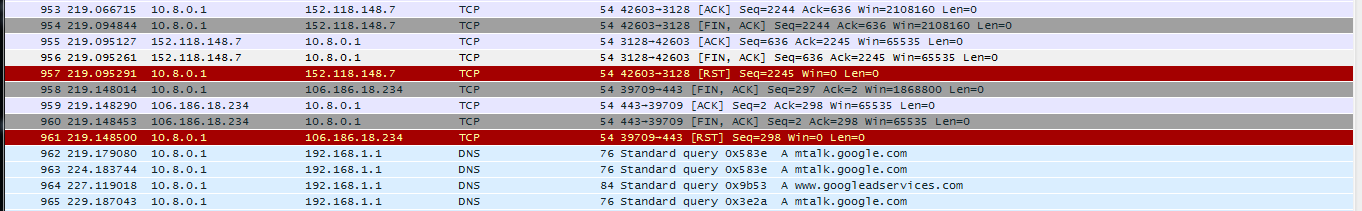
Tabel 4.3 Hasil perhitungan ∆t paket pada perangkat *laptop*

|  |  |
| --- | --- |
| **Pengambilan Data ke-** | **Delay (ms)** |
| 1 | 1053 |
| 2 | 1030 |
| 3 | 1013 |
| 4 | 1057 |
| **Rata-rata** | **1038,25** |

Tabel pertama menghasilkan 14 data minus dua data anomali, yang jika dihitung rata-ratanya telah memenuhi target *delay handoff* <100 ms. Ditambah PDU yang dievaluasi merupakan paket (PDU *layer* 3 OSI-*layer*) sehingga nilai tersebut dapat dikatakan waktu “kotor”. Sehingga dapat diperkirakan jika *handoff* yang terjadi kurang dari nilai tersebut. Namun kekurangan terletak pada banyak data yang mewakili hasil tersebut. Sehingga walaupun hasil yang didapatkan baik, penulis merasa untuk menyimpulkan bahwa metode ini merupakan solusi dari permasalahan *handoff* pada komunikasi multimedia secara *real-time* belum dapat terpenuhi.

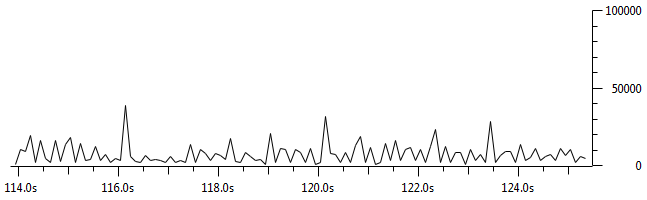
*Delay* yang didapat di tabel kedua merupakan jarak waktu (∆t) dari paket TCP terakhir yang diterima atau dikirim *laptop* hingga *laptop* mengirim *frame* DHCP mengingat proses *handoff* secara *default* disediakan oleh pengaturan DHCP[32]. Tabel kedua menghasilkan empat data dengan rata-rata *delay­* cukup tinggi. Analisis terhadap hasil ini adalah karena perangkat yang digunakan adalah *laptop* dengan NIC lebih baik dibanding perangkat *mobile,* paket dan *frame* yang dilewatkan lebih banyak sehingga memakan waktu lebih besar. Namun pengambilan data menggunakan *laptop* dianggap perlu untuk menggambarkan komunikasi *frame* pada proses *handoff* yang tidak terlihat dengan menggunakan tPacketCapture.

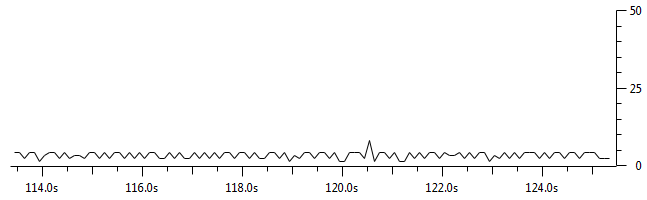
Contoh salah satu hasil pengambilan data sesuai parameter yang diharapkan adalah sebagai berikut. Gambar dibawah memperlihatkan rekaman paket TCP terakhir sebelum terjadi proses *handoff* hingga paket DNS pertama menuju ke AP baru. Koneksi Internet dengan perangkat hilang pada detik ke 219,095 dan terhubung ke OFAP2 pertama kali pada detik ke 219,179. Sehingga *delay* yang tercatat adalah 84 milidetik:



Gambar 4.8 Hasil *packet capture*

Grafik I/O yang diolah menggunakan salah satu fitur *Wireshark* juga menunjukkan kestabilan aliran data menggunakan jaringan berbasis *OpenFlow*:

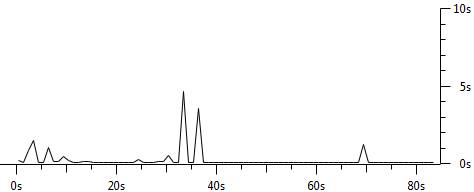


(a) Grafik waktu (sumbu x) terhadap besar paket dalam bytes (sumbu y)

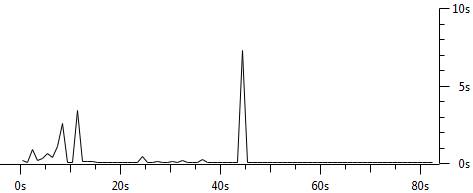
(b) Grafik waktu (sumbu x) terhadap total paket (sumbu y)

Gambar 4.9 Kondisi lalu lintas jaringan saat jaringan terhubung

Grafik diatas menunjukkan bahwa besar paket data (bytes) dan total paket yang diterima tidak pernah terputus (tidak pernah menyentuh angka 0). Hal ini menunjukkan bahwa kualitas aliran data lebih baik dan dapat diperkirakan bahwa perpindahan antara dua AP juga lebih baik. Pernyataan tersebut juga dapat dilihat dari grafik *jitter* di bawah ini:



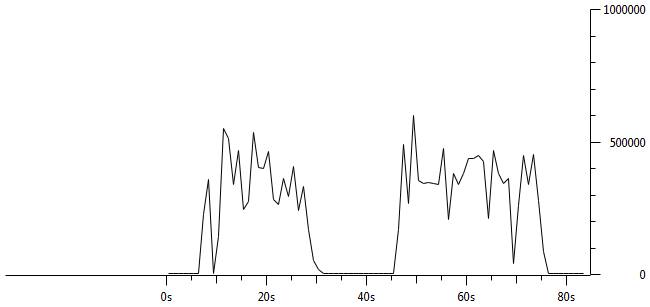
(a) *Jitter* pada saat *device* telah terasosiasi dengan kedua AP sebelumnya



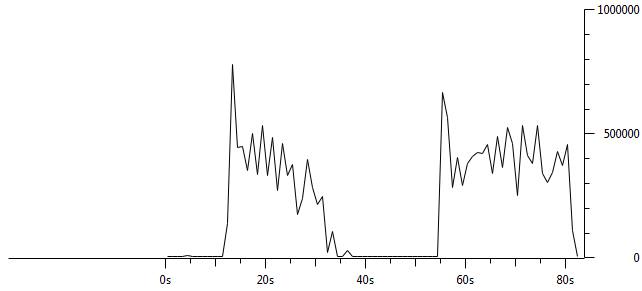
(b) *Jitter* pada saat *device* belum terasosiasi dengan kedua AP sebelumnya

Gambar 4.10 Grafik *jitter*

Dari data diatas didapatkan *delay* komunikasi TCP yang cukup besar (4 dan 8 detik) dimana perangkat yang sebelumnya telah terasosiasi dengan kedua AP memiliki *delay* yang lebih kecil. Naiknya grafik menggambarkan terjadinya penundaan dalam penerimaan paket, sehingga semakin kecil nilainya maka koneksi semakin baik. Grafik (b) diatas memperlihatkan kenaikan drastis di detik 44-46 dengan nilai 7-8 detik pada sumbu y dan (a) sebesar 4-5 detik. Dapat diartikan bahwa pada (b) paket diterima dengan interval 0 kemudian melambat hingga 7-8 detik dan kembali ke 0 lagi. Namun dapat dilihat pula grafik tersebut mayoritas bernilai 0 pada sumbu y jadi dapat ditarik kesimpulan bahwa koneksi tersebut stabil, begitu pula pada grafik (a). Data yang sama kemudian diolah untuk menggambarkan besar paket data yang diterima:



(a) Grafik penerimaan paket saat perangkat sudah terasosiasi sebelumnya dengan AP



(b) Grafik penerimaan paket saat perangkat sudah terasosiasi sebelumnya dengan AP

Gambar 4.11 Grafik besar paket yang diterima

Dapat dilihat dari gambar diatas bahwa kestabilan koneksi kembali ke level sebelum terjadi proses *handoff* dengan cepat. *Delaynya* yang tercatat mencapai 15 detik mengingat grafik *jitter* diatas hanya menggambarkan komunikasi TCP. Kedua gambar 4.10 dan 4.11 disertakan untuk menunjang pernyataan bahwa kestabilan koneksi berbasis *OpenFlow* lebih baik dibanding koneksi biasa. Analisis *delay* yang terjadi cukup besar ialah karena proses *handoff* terjadi di perangkat *mobile,* jika proses dilakukan di level kontroler maka *delay* akan jauh lebih kecil.

Sebagai tambahan dilakukan juga pengambilan data paket pada kontroler menggunakan *Wireshark.* Pengambilan data dilakukan sebanyak lima kali dengan kondisi: satu klien *mobile* yang terhubung ke salah satu OFAP melakukan koneksi terhadap kontroler melalui *streaming video,* kemudian klien tersebut berpindah ke OFAP lainnya. Kekurangannya adalah *Wireshark* yang digunakan untuk menangkap aliran paket tidak di-set sedemikian rupa untuk memilih *frame* mana saja yang akan dikirimkan sehingga hasil ∆t yang didapatkan saat kontroler melakukan koneksi ke satu OFAP kemudian berpindah hingga melakukan koneksi lagi ke OFAP yang lain tidak memuaskan. Protokol yang terekam adalah sebagai berikut:

Tabel 4.4 Protokol yang berjalan pada kontroler

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ARP | DB-LSP-DISC | ICMP, ICMPv6 | LLMNR | SRVLOC |
| AoE | DHCP, DHCPv6 | IGMPv1, IGMPv2, IGMPv3 | MDNS | SSDP |
| BJNP | DNS | LLC | NBNS | STP |
| BROWSER | HTTP | LLDP | OpenFlow | TCP |
| UDP | CDP | TLSv1.2 | SSHv2 |  |

Pada dasarnya hanya beberapa protokol yang diperlukan untuk menganalisis kualitas koneksi seperti DHCP untuk memonitor kualitas koneksi dengan tiap klien, OpenFlow, TCP, dan ARP untuk melihat berapa jumlah klien yang terhubung dan mobilitasnya. Ditambah dengan kontroler terhubung ke jaringan utama Departemen Teknik Elektro UI, sehingga terdapat beberapa *conversation* yang tidak dibutuhkan dan memakan waktu dalam perpindahan kontrol dengan perangkat diluar *testbed.* Karakteristik lalu lintas jaringan TCP dari kelima data yang didapat adalah sebagai berikut:

(a) Paket TCP terkirim antara OFAP1 dan server (b) Paket TCP terkirim antara OFAP2 dan server

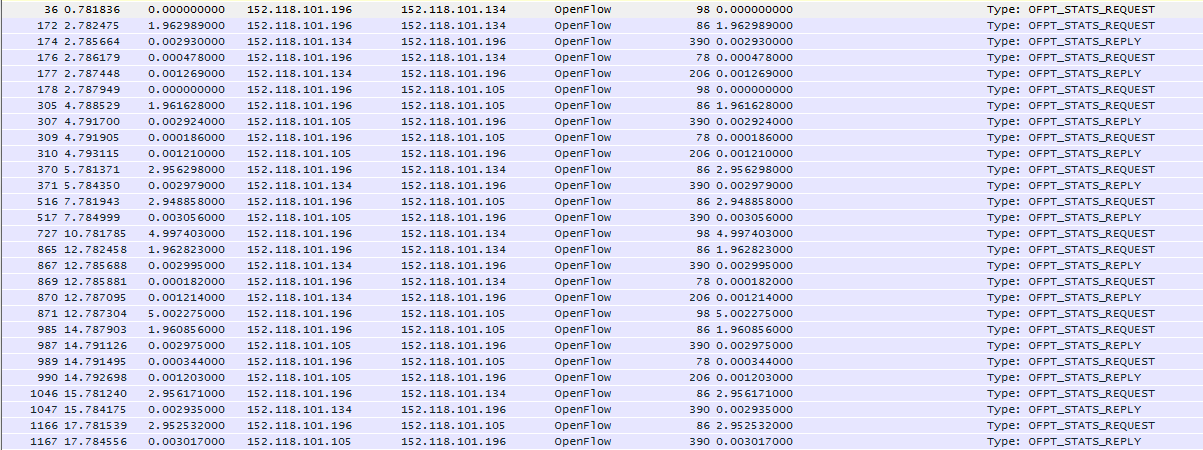
(c) Paket sesi OpenFlow antara OFAP1 dan server (d) Paket sesi OpenFlow antara OFAP2 dan server

Gambar 4.12 Grafik banyak paket yang diterima pada lalu lintas TCP

Secara umum, sebagian besar komunikasi berbasis TCP yang terjadi pada kontroler terbagi menjadi dua: paket data *streaming video* dari kontroler ke kedua OFAP menggunakan protokol TCP dan paket data sesi *OpenFlow* untuk *update* status koneksi kontroler dan OFAP menggunakan protokol *OpenFlow*. Hal ini dikarenakan saat pengambilan data dilakukan kondisinya adalah hanya ada satu koneksi *streaming video* dari perangkat *mobile* ke kontroler/server melalui kedua OFAP.

Grafik (a) memperlihatkan banyak paket yang terkirim antara OFAP1 dan Server tiap pengambilan data, grafik (b) menggambarkan banyak paket yang terkirim antara OFAP2 dan Server tiap pengambilan data, grafik (c) menggambarkan banyak paket *OpenFlow* antara OFAP1 dan Server tiap pengambilan data, dan grafik (d) menggambarkan banyak paket *OpenFlow* antara OFAP2 dan Server tiap pengambilan data. Jumlah paket memiliki perbedaan di setiap pengambilannya akibat perbedaan lama pengambilan data. Jendela karakterisasi tidak bisa disamakan karena tergantung kapan proses *handoff* terjadi, dimana setiap pengambilan data letak prosesnya berbeda. Terlihat pada komunikasi paket *OpenFlow* pada grafik (c) dan (d), paket yang dikirim dari server ke OFAP maupun sebaliknya adalah linear. Hal ini dikarenakan paket yang dikirim adalah paket REQUEST dan REPLY sehingga keduanya saling mempengaruhi. Anomali terjadi di pengambilan data pertama grafik (b) dimana tidak ada pengiriman paket *streaming* antara OFAP2 dan Server. Hal ini dikarenakan tidak ada koneksi yang terbangun antara perangkat *mobile* dengan OFAP saat pengambilan data tersebut dilakukan.

Paket *OpenFlow* yang terekam membuktikan bahwa jaringan ini menggunakan arsitektur SDN dalam operasinya. Gambar berikut adalah paket *OpenFlow* yang terekam dan kontennya berupa komunikasi kontroler dan OFAP untuk mempertahankan koneksi yang terbangun.



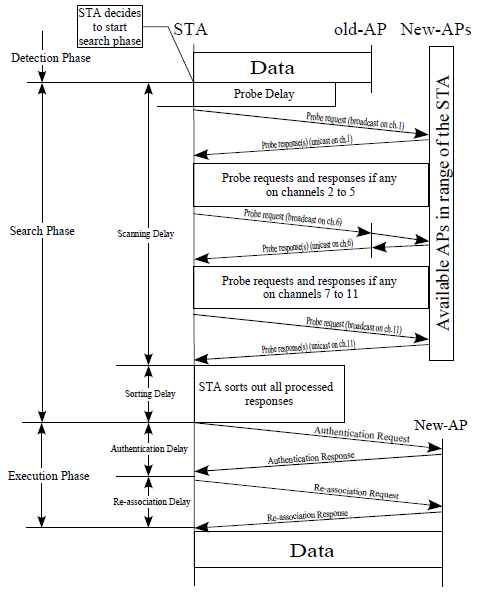
Gambar 4.13. *Screenshot* paket OpenFlow dalam aliran paket kontroler

Gambar diatas menunjukkan bahwa kontroler terus berkomunikasi dengan kedua OFAP (dengan alamat IP 152.118.101.134 dan 152.118.101.105) melalui pesan REQUEST dan REPLY.

Hasil yang beragam dan kurangnya data yang dapat diolah dalam riset ini diakibatkan oleh beberapa faktor. Beberapa faktor tersebut yang menjadi evaluasi riset ini adalah: (1) Fitur dari aplikasi yang digunakan untuk menangkap aliran data. Pertukaran *frame* dalam proses *handoff* tidak dapat ditampilkan karena data yang diambil hanyalah data dengan *header* di *layer* 3. Dari beberapa referensi dengan percobaan yang serupa, aplikasi yang digunakan merupakan hasil pengembangan riset itu sendiri, atau menggunakan perangkat *sniffer* khusus, atau menggunakan NIC khusus, dan sebagainya. (2) Ketidakstabilan *testbed.* Dalam 28 kali pengambilan data kondisi jaringan tidak selalu sama sehingga analisis yang dilakukan merupakan analisis yang dilakukan terhadap sebagian data, bukan seluruh data. (3) Topologi yang digunakan terlalu sederhana sehingga menyebabkan ketidakstabilan tersebut.

* 1. **Analisis Proses *Handoff***

Sesuai penjelasan pada bab II, proses *handoff* dibagi menjadi dua proses: *discovery* dan *re-authentication.*



Gambar 4.14 Proses *handoff*

Untuk menghitung total *delay* atau latensi proses *handoff,* sayangnya belum ada satu standar untuk digunakan. Namun penelitian ini menggunakan skema penghitungan *raw handoff latency.* Skema ini menghitung interval waktu sejak *probe* pertama yang diminta dari *client*/pengguna hingga respon *re-association* dari AP baru, ditambah *probe\_delay\_time.* Jadi rumusnya dapat dituliskan sebagai berikut:

Dimana Rhl adalah latensi total *raw handoff, ReAssResponse* adalah waktu dimana respon pertama pesan *re-association* diterima, *FirstProbeReq* untuk pesan *Probe Request* yang dikirimkan pertama kali dari *client*, dan *ProbeDelay* (*prob\_delay\_time*) adalah batas waktu bernilai tetap yang diberikan sebagai batas maksimal proses *discovery* di fase *search.* Fungsi ini diasumsikan telah diimplementasikan pada data ∆t tanpa adanya segmentasi *delay* secara jelas akibat kekurangan pada *tools packet capturer* yang digunakan. Rhl yang didapat pada perangkat *mobile* adalah sebesar 79,9 ms dan 1038,25 ms pada perangkat *laptop.*

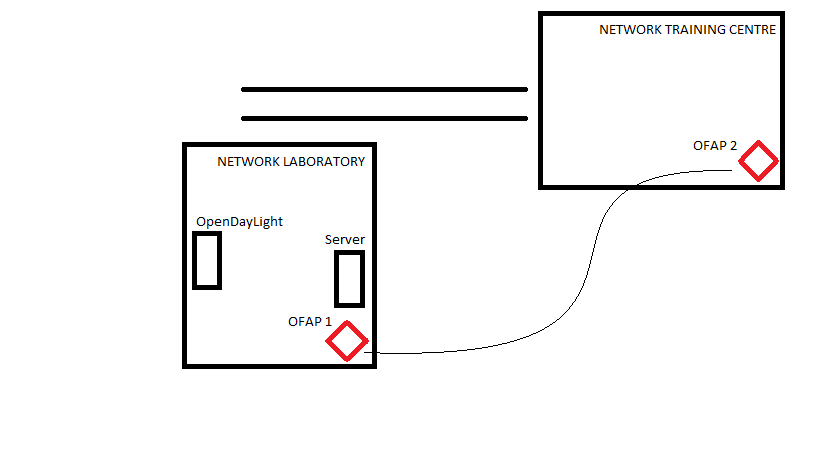
Dari penelitian yang dilakukan oleh [33], hasil *delay* pada proses *handoff* dalam jaringan *Wi-Fi* tradisional adalah sebagai berikut:

Tabel 4.5 Hasil pengukuran *delay handoff* pada [33]

|  |  |
| --- | --- |
| ***Delay Handoff* (ms)** | **Banyak Data** |
| 50-59 | 7 |
| 60-69 | 7 |
| 70-79 | 9 |
| 80-89 | 4 |
| 90-99 | 9 |
| 100-109 | 20 |
| 110-119 | 26 |
| 120-129 | 15 |
| 130-139 | 1 |
| 140-149 | 1 |
| 150-159 | 1 |
| **Rata-rata *Delay*** | **100,6 ms** |

Data diambil menggunakan *testbed* dimana kedua AP diletakkan menyerupai *testbed* dalam riset ini, dengan sinyal yang *overlap* seminimal mungkin namun perpindahan yang terjadi masih dapat diaplikasikan sebagai *fast handoff.* Riset pada [33] melakukan 100 kali iterasi pengambilan data. Data diatas menunjukkan rata-rata proses *handoff* membutuhkan waktu 100,6 ms. Sehingga dengan hasil Rhl yang menunjukkan hasil perhitungan *delay* menunjukkan hasil yang lebih baik dibanding rata-rata *delay* jaringan *Wi-Fi* tradisional, meskipun belum dapat merepresentasikan keabsahan teorinya akibat data yang belum mencukupi, dapat dipastikan bahwa jaringan *Wi-Fi* berbasis *OpenFlow* dapat dijadikan skema pengurangan waktu *delay* proses *handoff* dan berpotensi untuk diimplementasikan secara luas sekaligus untuk mendapatkan keuntungan fleksibilitas, kontrol, dan *cost* yang lebih rendah dibandingkan implementasi jaringan tradisional.

* 1. **Struktur *Test bed***



Distance ≈ 7 m

Gambar 4.15 Denah penempatan perangkat jaringan

Posisi kedua OFAP dipastikan tidak menghasilkan interferensi sinyal antara kedua OFAP namun jaraknya keduanya dibuat seminimal mungkin, sehingga tidak ada kondisi dimana perangkat tidak terhubung ke salah satu dari kedua OFAP tersebut.

# BAB 5 KESIMPULAN

# Kesimpulan

# Rata-rata *delay* proses *handoff* dalam jaringan *Wi-Fi* berbasis *OpenFlow* menggunakan skema *fast handoff* adalah sebesar 79,9 milisekon dengan menggunakan perangkat *mobile* dengan instalasi *tools packet capturer* sebagai alat ukurnya.

1. Rata-rata *delay* pada proses *handoff* dalam jaringan *Wi-Fi* berbasis *OpenFlow* menggunakan skema *fast handoff* adalah sebesar 1038,25 ms akibat perbedaan pengaturan stack yang diaktifkan di aplikasi packet capturer pada perangkat mobile yang digunakan. Sehingga menyebabkan beragamnya frame dan paket yang dilewatkan di lalu lintas jaringan dan berimbas pada delay.
2. Data yang dihasilkan belum sepenuhnya menjadikan bahwa jaringan berbasis *OpenFlow* dapat digunakan sebagai solusi *delay handoff*pada komunikasi multimedia *real-time.*Hal ini dikarenakan *tools*yang digunakan untuk menangkap PDU tidak memenuhi fungsi yang diperlukan dalam proses evaluasi PDU untuk mendapatkan nilai proses *handoff*, topologi yang digunakan terlalu sederhana, dan *flow*yang dirumuskan belum menunjang pengurangan waktu *delay*proses *handoff.*
3. Namun dari data tersebut, dapat dilihat bahwa arsitektur ini memiliki potensi untuk menjadi solusi pengurangan total waktu *delay handoff*.
4. Selain sebagai solusi besarnya *delay* proses *handoff* pada jaringan *wireless,* implementasi *OpenFlow*pada jaringan *Wi-Fi* dapat dikembangkan untuk meningkatkan fleksibilitas jaringan dan mengurangi *cost* serta mengubah proses *handoff*tradisional yang berbasis *client*(*client initiate*) menjadi *network initiate.*
5. **Saran**

Terkait dengan kekurangan yang dijelaskan di bagian sebelumnya, jika penelitian ini ingin diteruskan untuk mendapatkan hasil pembuktian pengurangan nilai *delay* proses *handoff* menggunakan arsitektur SDN terdapat beberapa poin yang bisa diajukan:

1. Penggunaan jaringan tanpa *proxy.* Jika penelitian akan dijalankan menggunakan jaringan kampus, alamat IP yang akan diberikan ke perangkat sebaiknya merupakan alamat IP di segmen DMZ.
2. *Tools* untuk merekam paket lebih baik. Dari beberapa penelitian dengan parameter serupa, *tools* yang digunakan untuk merekam paket merupakan pengembangan tim penelitian sendiri, atau menggunakan perangkat *sniffer* tambahan, atau menggunakan *Network Interface Card* (NIC) khusus. Penelitian ini masih menggunakan *tools* yang didistribusikan sehingga hasil yang diharapkan belum mampu didapat.
3. Dikerjakan secara tim. Komponen penelitian ini cukup banyak sehingga membutuhkan lebih dari satu orang peneliti.
4. Perubahan topologi. Topologi yang digunakan terlalu sederhana. Diantara kontroler dan OFAP seharusnya diletakkan *switch* berbasis *OpenFlow* dengan koneksi Ethernet ke kontroler sehingga kontrol yang dilakukan kepada OFAP akan lebih stabil.
5. Perumusan *flow* yang lebih baik. Penambahan *flow* menggunakan kontroler *OpenDayLight* memang paling mudah dan sederhana ditambah dengan fitur *user interface,* namun parameter yang dibutuhkan belum dapat dipenuhi. Perubahan dapat dilakukan dengan mengubah kontroler yang digunakan (contohnya, ProGFE) atau mengoperasikan *OpenDayLight* melalui *command line* sehingga konfigurasi parameter dapat dilakukan lebih leluasa.

**DAFTAR PUSTAKA**

1. C. Conner (2013) Fifty Essential Mobile Marketing Facts. [Online]. <http://www.forbes.com/sites/cherylsnappconner/2013/11/12/fifty-essential-mobile-marketing-facts/>
2. Maeve Duggam & Aaron Smith, Cell Internet Use, 2013, Pew Research Internet Project, Sept. 16, 2013, at p.7 ("among cell owners"). http://www.pewinternet.org/2013/09/16/cell-internet-use-2013/ and U.S. Census, Annual Estimates of the Resident Population by Single Year of Age and Sex for the United States: April 1,2010 to July 1,2013, 2013 Population Estimates, at <http://factfinder2.census.gov/faces/tableservices/jsf/pages/productview.xhtml?src=bkmk>.
3. V. Afshar (2014) 50 Incredible WiFi Tech Statistics That Businesses Must Know. [Online]. <http://www.huffingtonpost.com/vala-afshar/50-incredible-wifi-tech-s_b_4775837.html>
4. Vangie Beal (2014) Wi-Fi. [Online]. <http://www.webopedia.com/TERM/W/Wi_Fi.html>
5. A. Mishra, M. Shin, and W. Arbaugh, “An Empirical Analysis of the IEEE 802.11 MAC Layer Handoff Process,” ACM SIGCOMM Comp. Commun. Review, vol. 33, no. 2, Apr. 2003, pp. 93–102.
6. S. Shin et al., “Reducing MAC Layer Handoff Latency in IEEE 802.11 Wireless LANs,” Proc. ACM MobiWac 2004, Oct. 2004.
7. P. Bahl, R. Chandra, and J. Dunagan, “SSCH: Slotted Seeded Channel Hopping for Capacity Improvement in IEEE 802.11 Ad-Hoc Wireless Networks,” Proc. ACM Mobicom 2004, Oct. 2004.
8. H. Yang et al., “Securing a Wireless World,”Proc. IEEE, vol. 94, no. 2, Feb. 2006, pp. 442–54.
9. A. Mishra et al., “Proactive Key Distribution Using Neighbor Graphs,” IEEE Wireless Commun. Mag., vol. 11, no. 1, Feb. 2004, pp. 26–36.
10. Y. Choi et al., “Enhancement of a WLAN-Based Internet Service,” ACM Mobile Networks and Applications, vol. 10, no. 3, June 2005, pp. 303–14.
11. IEEE 802.11f, “Recommended Practice for Multi-Vendor Access Point Interoperability via an Inter-Access Point Protocol Across Distribution Systems Supporting IEEE 802.11 Operation,” IEEE Standard, July 2003.
12. H. Velayos and G. Karlsson, “Techniques to Reduce IEEE 802.11b MAC Layer Handover Time,” Proc. IEEE ICC 2004, June 2004.
13. I. Ramani and S. Savage, “SyncScan: Practical Fast Handof for 802.11 Infrastructure Networks,” Proc. IEEE Infocom 2005, Ma. 2005.
14. V. Brik, V. Mishra, and S. Banerjee, “Eliminating Handoff Latencies in 802.11 WLANs using Multiple Radios: Aplications, Experience, and Evaluation,” Proc. ACM Internet Measurement Conf. 2005, Oct. 2005.
15. K. Chi, J. Jiang, and L. Yen, “Cost-Effective Caching for Mobility Support in IEEE 802.1X Frameworks,” IEEE Trans. Mobile Computing, vol. 5, no. 1, Nov. 2006, pp. 1547–60.
16. M. Bargh et al., “Fast Authentication Mothods for Handovers between IEEE 802.11 Wireless LANs,” Proc. ACM WMASH 2004, Oct. 2004.
17. S. Pack and Y. Choi, “Fast Handoff Scheme based on Mobility Prediction in Public Wireless LAN Systems,” IEE Proc. Commun., vol. 151, no. 5, Oct. 2004, pp. 489–95.
18. IEEE 802.1x, “IEEE Standards for Local and Metropolitan Area Networks: Port based Network Access Control,” IEEE Standard, June 2001.
19. D. Schwab and R. Bunt, “Characterising the Use of a Campus Wireless Network,” Proc. IEEE INFOCOM 2004, Mar. 2004.
20. A. Mishra, M. Shin, and W. Arbaugh, “Context Caching using Neighbor Graphs for Fast Handoffs in a Wireless Network,” Proc. IEEE INFOCOM 2004, Mar. 2004.
21. J. Loughney et al., “Context Transfer Protocol (CXTP),” IETF RFC 4067, July 2005.
22. A. Balachandran et al., “Characterizing User Behaviour and Network Performance in a Public Wireless LAN,” Proc. ACM SIGMETRIC 2002, June 2002.
23. S. Pack et al., “SNC: A Selective Neighbor Caching Scheme for Fast Handoff in IEEE 802.11 Wireless Networks,” ACM Mobile Computing and Commun. Review, vol. 9, no. 4, Oct. 2005, pp. 39–49.
24. Software-Defined Networking: The New Norm for Networks ONF White Paper April 13, 2012
25. (2013) Software Defined Networking. [Online]. <http://theinstitute.ieee.org/benefits/ieee-groups/softwaredefined-networks-explained>
26. N. McKeown, T. Anderson, H. Balakrishnan, G. M. Parulkar, L. L. Peterson, J. Rexford, S. Shenker, and J. S. Turner, “OpenFlow: enabling innovation in campus networks,” Computer Communication Review, vol. 38, no. 2, pp. 69–74, 2008.
27. (2014) Install Java JDK. [Online]. <http://askubuntu.com/questions/56104/how-can-i-install-sun-oracles-proprietary-java-jdk-6-7-8-or-jre/173951#173951>.
28. (2014) Table of Hardwere. [Online]. <http://wiki.openwrt.org/toh/start>
29. (2011) Pantou : OpenFlow 1.0 for OpenWRT. [Online]. <http://archive.openflow.org/wk/index.php/Pantou_:_OpenFlow_1.0_for_OpenWRT>
30. (2012) About tPacketCapture. [Online]. http://www.taosoftware.co.jp/en/android/packetcapture/ diakses tgl 14 Des
31. D. Murray, M. Dixon, and T. Koziniec, "Scanning Delays in 802.11 Networks," The 2077 Internation Conference on Next Generation Mobile Applications, Services, and Technologies, June 2007
32. Y. Amir, C. Danilov, M. Hilsdale, R. Musaliou-Elefteri, and N. Rivera, “Fast Handoff for Seamless Wireless Mesh Networks,” MobiSys'06, June 19-22, 2006, Uppsala, Sweden.
33. R. Corvaja, A. Zanella, M. Dossi, A. Tontoli, and P. Zennaro, “Experimental of the Handover Procedure in a WiFi Network,” WPMC04, Sept 2004.

Lampiran

Dalam merancang *Testbed,* terdapat beberapa komponen utama untuk membangun topologi ini:

1. Instalasi Quagga

Quagga yang digunakan adalah Quagga versi 0.99.20.1 dengan langkah-langkah sebagai berikut:

* Instalasi dari sumber

quagga@quagga-server:~$sudo apt-get install quagga quagga-doc

* Mengedit file *daemon* sebagaikonfigurasi protokol *routing* yang diaktifkan melalui pernyataan yes/no pada file daemons

quagga@quagga-server:~$sudo gedit /etc/quagga/daemons

* Yang diaktifkan adalah zebra dan *daemon* ospf, sehingga konfigurasinya adalah sebagai berikut:

zebra=yes

bgp=no

ospf=yes

ospf6d=no

ripd=no

ripngd=no

* Membuat file konfigurasi zebra.conf dan ospf.conf dengan menyalin contoh file konfigurasi yang disediakan

quagga@quagga-server:~$sudo cp /usr/share/doc/quagga/examples/zebra.conf.sample /etc/quagga/zebra.conf

quagga@quagga-server:~$sudo cp /usr/share/doc/quagga/examples/ospf.conf.sample /etc/quagga/ospf.conf

* Mengubah kepemilikan file terhadap *user* dan *group* dalam direktori /etc/quagga

quagga@quagga-server:~$sudo chown quagga.quaggavty /etc/quagga/\*.conf

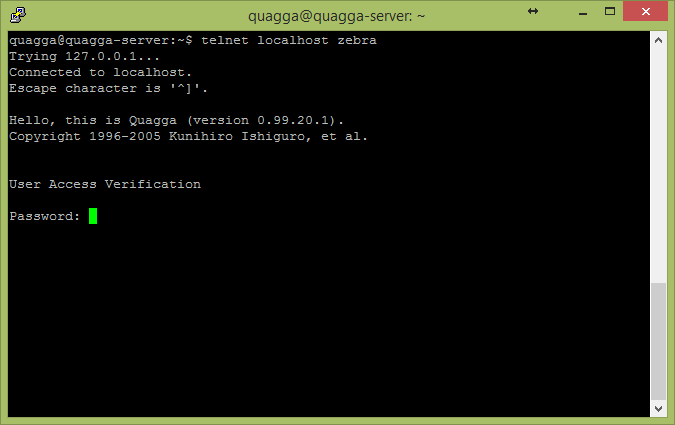
quagga@quagga-server:~$sudo chmod 640 /etc/quagga/\*.conf

* Start/stop daemons

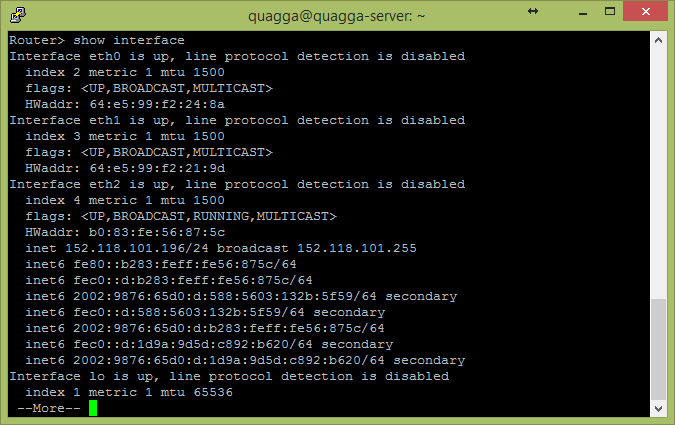
quagga@quagga-server:~$sudo /etc/init.d/quagga start/stop/restart

* Jika berhasil, quagga dapat di-telnet menggunakan perintah

quagga@quagga-server:~$telnet localhost zebra



Gambar 4.1 *Screenshot* quagga telah aktif



Gambar 4.2 *Screenshot* saat *interface* sudah terhubung dengan quagga dimana alamat IP 152.118.101.196 adalah IP server

1. Instalasi OpenDayLight

Untuk dapat menjalankan OpenDayLight, Java sudah harus diinstall di Ubuntu[27]. Untuk memulai instalasi OpenDayLight:

* Mengunduh *release package* dari http://www.opendaylight.org/software/downloads dan memilih versi OpenDayLight yang ingin digunakan. Penulisan ini menggunakan Hydrogen Base.
* *Un-zip* file yang diunduh, kemudian masuk ke folder tersebut
* Mengksekusi perintah ./run.sh or ./run.bat dengan peran administrator (sudo) untuk menjalankan OpenDayLight
* Membuka aplikasi perambah web dan navigasi ke http://<alamat-ip-perangkat-tempat-menjalankan-OpenDayLight>:8080 untuk membuka *web interface*
* Halaman pertama akan meminta *username* dan *password* yang secara *default* adalah admin:admin
* Jika OpenDayLight dijalankan di perangkat yang sama, dapat menggunakan <http://localhost:8080> atau <http://127.0.0.1:8080>
* Jika ingin menjalankan OpenDayLight di *background*  dapat menggunakan perintah ./run.sh –start (sudo) atau ./run.sh –stop untuk menghentikan
* Perintah ./run.sh –status dapat dieksekusi untuk melihat apakah aplikasi aktif atau tidak

1. *Build* Pantou

Perangkat keras yang digunakan adalah Buffalo WZR-HP-G450H, sehingga untuk membuat OpenWRT harus disesuaikan apakah *adjustment* yang digunakan adalah *trunk, attitude,* atau *backfire*[28]*.* Perangkat keras yang digunakan dibangun dengan *adjustment attitude.* Langkah pertama dalam membuat *access point* berbasis OpenFlow ini (OFAP) adalah dengan membuat *image* OpenWRT-OpenFlow 1.0 dengan *adjustment attitude*[29]*.*

* Mempersiapkan komponen yang dibutuhkan

apt-get install build-essential binutils flex bison autoconf gettext texinfo \ sharutils subversion libncurses5-dev ncurses-term zlib1g-dev gawk

* Mengunduh *attitude*

Mkdir ~/ofwrt

cd ~/ofwrt

svn co svn://svn.openwrt.org/openwrt/branches/attitude\_adjustment

cd attitude\_adjustment

./scripts/feeds update -a

./scripts/feeds install -a

* Mengunduh paket OpenFlow 1.0

cd ~/ofwrt/

git clone git://gitosis.stanford.edu/openflow-openwrt

git checkout –b openflow-1.0/tplink origin/openflow-1.0/tplink

cd ~/ofwrt/attitude\_adjustment/packages/

ln -s ~/ofwrt/openflow-openwrt/openflow-1.0/

cd ~/ofwrt/attitude\_adjustment/

ln -s ~/ofwrt/openflow-openwrt/openflow-1.0/files

make menuconfig

* Memilih sistem yang sesuai pada menuconfig (AR71xx untuk perangkat keras yang digunakan)
* Memilih paket ‘OpenFlow’ dalam sub pilihan ‘network’
* Memilih paket ‘tc’ dalam sub pilihan ‘network’
* Memilih ‘kmod-tun’ dalam sub pilihan Kernel Modules->Network Support
* Keluar dan simpan
* Untuk menambahkan fitur untuk *queueing*

Make kernel\_menuconfig

* Dalam sub pilihan Networking Support->Networking options->QoSdan/atau fair queueing pilih Hierarchical Token Bucket (HTB)
* File diimpan kemudian keluar
* Buat file imagenya

make prereq

make V=99

* *File image* yang telah ter-*build* kemudian dilanjutkan dengan mem-*flash* *file* tersebut pada perangkat

Langkah kedua adalah memasukkan konfigurasi OpenFlow 1.0 ke OpenWRT

* Menentukan *username* dan *password* untuk tiap OFAP. Penelitian ini menggunakan root:admin
* Memasukkan konfigurasi file /etc/config/network untuk AP1 sebagai berikut:

config interface 'loopback'

option ifname 'lo'

option proto 'static'

option ipaddr '127.0.0.1'

option netmask '255.0.0.0'

config interface 'lan'

option ifname 'eth0.1'

option type 'bridge'

option proto 'static'

option ipaddr '192.168.1.1'

option netmask '255.255.255.0'

config interface 'wan'

option ifname 'eth0.2'

option proto 'dhcp'

config switch

option name 'switch0'

option reset '1'

option enable\_vlan '1'

config switch\_vlan

option device 'switch0'

option vlan '1'

option ports '0t 2 3 4 5'

config switch\_vlan

option device 'switch0'

option vlan '2'

option ports '0t 1'

* Memasukkan konfigurasi file /etc/config/network untuk AP2 sebagai berikut:

config interface 'loopback'

option ifname 'lo'

option proto 'static'

option ipaddr '127.0.0.1'

option netmask '255.0.0.0'

config interface 'lan'

option ifname 'eth0.1'

option type 'bridge'

option proto 'static'

option ipaddr '192.168.2.1'

option netmask '255.255.255.0'

config interface 'wan'

option ifname 'eth0.2'

option proto 'dhcp'

config switch

option name 'switch0'

option reset '1'

option enable\_vlan '1'

config switch\_vlan

option device 'switch0'

option vlan '1'

option ports '0t 2 3 4 5'

config switch\_vlan

option device 'switch0'

option vlan '2'

option ports '0t 1'

* Memasukkan konfigurasi file /etc/config/openflow untuk AP1 sebagai berikut:

config 'ofswitch'

option 'dp' 'dp0'

option 'dpid' '000000000001'

option 'ofports' 'eth0.1 eth0.2 wlan0'

option 'ofctl' 'tcp:152.118.101.196:6633'

option 'mode' 'outofband'

* Memasukkan konfigurasi file /etc/config/openflow untuk AP2 sebagai berikut:

config 'ofswitch'

option 'dp' 'dp0'

option 'dpid' '000000000002'

option 'ofports' 'eth0.1 eth0.2 wlan0'

option 'ofctl' 'tcp: 152.118.101.196:6633'

option 'mode' 'outofband'

* Memasukkan parameter –no-slicing ke /lib/openflow/ofswitch.sh (AP1 dan AP2)

#!/bin/sh

# Copyright (C) 2006 OpenWrt.org

#DEBUG="echo"

add\_ofswitch\_datapath() {

local config="$1"

local ofports

local dpports

local dp

local mode

local dpid

config\_get ofports "$config" ofports

config\_get dp "$config" dp

config\_get mode "$config" mode

config\_get dpid "$config" dpid

dpports=`echo "$ofports" | tr ' ' ','`

echo "$dpports"

pidfile="/var/run/ofdatapath.pid"

[ -n "$dpports" ] && {

if [[ "$mode" == "inband" ]]

then

echo "Configuring OpenFlow switch for inband control"

[ -n "$dpid" ] && {

ofdatapath punix:/var/run/dp0.sock -i "$dpports" --no-slicing --local-port=tap:tap0 "--pidfile=$pidfile" -d "$dpid" &

} || {

ofdatapath punix:/var/run/dp0.sock -i "$dpports" --no-slicing --local-port=tap:tap0 "--pidfile=$pidfile" &

}

else

echo "Configuring OpenFlow switch for out-of-band control"

[ -n "$dpid" ] && {

ofdatapath punix:/var/run/dp0.sock -i "$dpports" --no-slicing --no-local-port "--pidfile=$pidfile" -d "$dpid" &

} || {

ofdatapath punix:/var/run/dp0.sock -i "$dpports" --no-slicing --no-local-port "--pidfile=$pidfile" &

}

fi

}

}

setup\_ofswitch() {

local config="$1"

add\_ofswitch\_datapath "$config"

}

* Melakukan *restart* padasemua servis dengan perintah /etc/init.d/firewall, /etc/init.d/dnsmasq, /etc/init.d/network
* Menjalankan servisOpenFlow dengan perintah /etc/init.d/openflow start

1. Asosiasi mengacu pada prosedur untuk membuat koneksi logika oleh MH yang akan dijalankan saat MH masuk ke sebuah sistem WLAN untuk pertama kalinya. Di sisi lain, prosedur re-asosiasi adalah prosedur saat MH menyediakan informasi kepada sistem WLAN yang berisi keterangan dimana MH sebelumnya terhubung [↑](#footnote-ref-1)