Simulasi Mobility pada Software Defined Networking

Baik Budi, Sigit Haryadi
Institut Teknologi Bandung
Jl. Ganesha No.10, Bandung, Jawa Barat, (022) 2500935
e-mail: baikbudi@students.itb.ac.id, sigit@stei.itb.ac.id

Abstrak – *Software Define Networking adalah sebuah arsitektur yang dinamis, dapat dikelola, hemat biaya, dan mudah beradaptasi, sehingga ideal untuk bandwidth yang tinggi, dan bersifat dinamis untuk aplikasi dan layanan saat ini. Arsitektur software defined networking ini adalah dengan memisahkan control plane dan data plane. Saat ini Software Defined Wireless Networking merupakan paradigma baru dalam jaringan nirkabel, yang secara fisik memisahkan control plane dan data plane dari berbagai macam elemen infrastruktur wireless. Pada skenario mobility dengan topologi Tree dengan menggunakan controller POX telah berhasil dilakukan skenario Mobiity. Masing-masing station berhasil terhubung kembal ke access point yang baru setelah sebelumnya berada diluar jangkauan access point yang lama ketika proses mobility berlangsung, yang sempat terputus selama 4 detik sebelum terhubung kembali ke access poin yang baru. Dari penelitian ini dapat dilihat bahwa controller POX dapat meng-handle proses Mobility. Pada skenario mobility dengan topologi Tree, menggunakan controller POX, waktu tercepat yang dibutuhkan untuk terkoneksi dengan access point yang baru adalah 4 detik.*

Kata kunci*: Software Defined Networking, Software Defined Wireless Networking, Open vSwitch, POX controller, Moblility*

1. Pendahuluan

SDN *(Software Defined Networking)* merupakan sebuah arsitektur yang dinamis, dapat dikelola, hemat biaya, dan mudah beradaptasi, sehingga ideal untuk *bandwidth* yang tinggi, bersifat dinamis untuk aplikasi dan layanan saat ini. Arsitektur SDN ini adalah dengan memisahkan *control plane* dan *data plane* [1]. Konsep utama pada SDN adalah sentralisasi jaringan dengan semua pengaturan berada pada *control plane.* Konsep SDN ini sangat memudahkan operator atau administrator jaringan dalam mengelola jaringannya. SDN juga mampu memberikan solusi untuk permasalahan-permasalahan jaringan yang sekarang seperti sulitnya mengintegrasikan teknologi baru karena alasan perbedaan perangkat atau platform, kinerja yang buruk karena ada beberapa operasi yang berlebihan pada *protocol layer* dan sulitnya menyediakan layanan-layanan baru.

Baru-baru ini juga diperkenalkan sebuah konsep yang dikenal dengan nama SDWN *(Software Defied Wireless Networking).* Konsep ini merupakan konsep SDN yang digunakan pada jaringan nirkabel untuk menambah fleksibilitas dengan meningkatnya lalu lintas data melalui jaringan nirkabel tersebut. SDWN juga merupakan cabang penelitian baru dari SDN [2], didasari juga oleh meningkatkan perhatian operator seluler dalam bidang ini [3]. Beberapa tahun terakhir sudah muncul beberapa penelitian yang membahas tentang penerapan SDN untuk jaringan nirkabel [4], [5], [6], [7]. Dengan adanya kontrol terpusat pada SDWN akan memberikan kemudahan pada administrator jaringan untuk mengelola dan melakukan manajemen trafik dalam jaringan nirkabel.

Ketika akan melakukan eksperimen SDN dan protokol OpenFlow dalam jaringan nirkabel, hanya ada beberapa alternatif dalam pemilihan simulator yang akan digunakan melakukan simulasi SDWN tersebut, seperti OMNET++ [5], NS3, dan Estinet. SDN simulator seperti

Mininet [8] tidak mendukung pemodelan dengan *wireless channel.* Mininet sebagai simulator SDN yang juga menjadi dasar untuk prototype, pengujian dan *benchmark* dari protokol baru, serta pembuatan arsitektur jaringan *end to end* dan aplikasi.

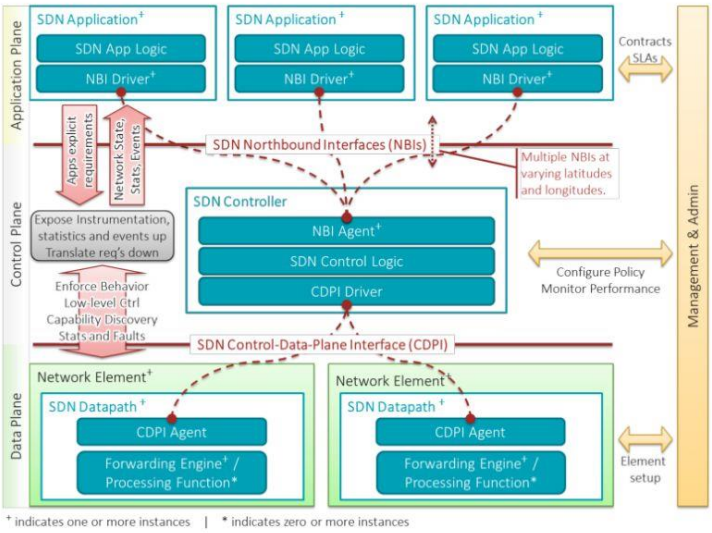
Pada paper ini akan dibahas mengenai simulasi SDN menggunakan Mininet yang telah ditambahkan ekstensi Mininet-WiFi, sehingga dengan memanfaatkan ekstensi ini bisa ditambahkan *access point* dan *station*. Dengan penambahan *access point*, maka dalam topologi SDN bisa ditambahkan *host* yang terhubung secara *wireless* ke *Open vSwitch* melalui *access point*. Pada penelitian ini juga dibahas mengenai *mobility* dalam SDN, dimana dalam skenario yang dibuat, *user* atau *host* akan bergerak dari satu akses poin ke akses poin lain. *Host* yang sebelumnya terhubung dengan sebuah akses poin akan terhubung dengan akses poin lainnya ketika berada di jangkauan akses poin tersebut. *Open vSwich controller* yang digunakan adalah POX. Pada skenario ini akan dilihat apakah POX *controller* mampu menangani skenario *Mobility* tersebut.

2. Software Define Networking

Kebanyakan perangkat jaringan saat ini mempunyai fungsi operasi kontrol dan aliran data dalam perangkat yang sama. Satu-satunya kontrol yang tersedia untuk administrator jaringan adalah dalam *network management plane,* yang digunakan untuk mengkonfigurasi setiap *node* jaringan secara terpisah. Sifat statis cari perangkat jaringan saat ini tidak memungkinkan konfigurasi *control plane* secara penuh. Adapun tujuan utama SDN [9] adalah menyediakan manajemen kontrol bagi pengguna untuk mengatur *forwarding hardware* dari elemen jaringan. Adapun gagasan dibalik munculnya SDN adalah sentralisasi *control plane,* dan memisahkan *data plane* dari *control plane.* Dengan demikian, perangkat keras jaringan tetap menggunakan *switching fabric (data plane),* tetapi menyerahkan fungsi *intelligence (switching* dan *routing information)* kepada *controller*. Dengan hal ini memungkinkan administrator jaringan unruk mengkonfigurasi elemen jaringan langsung dari controller, dimana dengan kontrol terpusat ini akan membuat jaringan menjadi sangat fleksibel.

2.1 Arsitektur SDN

Arsitektur SDN di standarisasi oleh ONF *(Open Network Foundation)* [10]. SDN dirancang dengan empat fitur kunci: Memisahkan *control plane* dan *data plane;* kontrol terpusat SDN *(controller)*; *Open Interface* antara *control plane (controller)* dengan *data plane;* dan kemampuan memprogram jaringan melalui aplikasi eksternal. Karakteristik tersebut membuat SDN memimpin dalam meningkatkan dukungan terhadap jaringan di masa depan. Struktur SDN dapat dlihat dari gambar 1 dibawah ini:



Gambar 1. Arsitektur SDN [11]

Arsitektur SDN dapat dibagi menjadi 3 layer [11]:

1. *Infrastructure layer*
   Terdiri dari elemen jaringan yg dapat mengatur *SDN Datapath* sesuai dengan instruksi yg diberikan melalui CDPI *(Control-Data-Plane Interface).*
2. *Control layer*
   Entitas kontrol (*SDN Controller*) yang mentranslasikan kebutuhan aplikasi dengan infrastruktur dengan memberikan instruksi yg sesuai untuk *SDN Datapath* serta memberikan informasi yg relevan dan dibutuhkan oleh *SDN Application.*
3. *Application layer*
   Berada pada lapis teratas, berkomunikasi dengan sistem via NBI *(Northbound Interface).*

2.2 *OpenFlow*

OpenFLow merupakan protokol yang digunakan untuk mengelola *Southbound Interface* dalam arsitektur SDN. OpenFlow adalah standar *interface* pertama yang ditetapkan untuk memfasilitasi interaksi anatar *control plane* dan *data plane* dalam arsitektur SDN. OpenFlow menyediakan akses berbasis *software* untuk *flow table* dalam menginstruksikan *switch* dan *router* bagaimana mengarahkan lalu lintas trafik dalam jaringan. Dengan menggunakan *flow table*, administrator dapat dengan cepat mengubah *layout* jaringan dan lalu lintas jaringan. OpenFlow juga menyediakan sebuah *management tool* yang dapat digunakan untuk mengontrol beberapa fitur, seperti perubahan topology dan *packet filtering.*

Spesifikasi dan standar dari OpenFLow ditentukan dan dikontrol oleh ONF, yang dipimpin oleh seorang direksi dari 7 perusahaan yang memiliki da mengoperasikan bebapa jaringan terbesar di dunia, yaitu *Verizon, Google, Microsoft, Facebook, Deutsche Telekom, NTT* dan *Yahoo.* Beberapa vendor perangkat keras terkemuka, seperti IBM, HP, dan Cisco telah menawarkan beberapa *switch* dan *router* yang telah mendukung protokol OpenFLow. Terdapat dua jenis OpenFlow *switch*, yaitu OpenFLow-Only dan OpenFlow-Hybrid, dimana dalam OpenFlow-only, semua paket diolah oleh pipa OpenFlow, sedangkan OpenFlow-hybrid, dalam *switch* tersebut mendukung operasi OpenFLow-only dan juga operasi *switching* Ethernet normal (tradisional L2 dan L3 *switching* dan *routing*) [12].

Arsitektur OpenFlow terdiri dari berbagai potongan peralatan *switching* OpenFlow yang dikelola oleh satu atau lebih *controller* OpenFLow. Koneksi antara OpenFlow controller dengan *switch* menggunakan SSL atau TLS *cryptographic*, dimana *switch* dan *controller* keduanya di autentikasi dengan cara pertukaran *certificates* yang telah di *signed* oleh kedua belah pihak. Meskipun telah dilengkapi dengan algoritma security yang *powerful*, controller tersebut masih rentan terhadap serangan *(Denial of Service)*, oleh karena itu perlu diimplementasikan *security* yang tepat menghindari serangan yang mungkin akan terjadi.

2.3 *OpenFlow Switch*

OpenFlow *switch* merupakan perangkat *forwarding* yang meneruskan paket berdasarkan *flow table.* OpenFlow *switch* mirip dengan perangkat *forwarding* tradisional, tetapi *flow table* tidak dikelola oleh *switch*, melainkan terhubung dengan *controller* melalui *secure channel* antara *switch* dengan *controller.*

OpenFlow *switch* dibagi dalam dua kategori, yaitu *Commercial Switch* dan *Software Switch. Commercial Switch* adalah *switch* fisik dengan perangkat keras yang telah mendukung OpenFlow. Beberapa vendor yang menawarkan *Commercial Switch* yang telah mendukung OpenFlow antara lain Cisco, HP, dan NEX. Sedangkan *Software Switch* adalah perangkat lunak yang telah mendukung OpenFlow dan bisa diinstal dalam sebuah *general purpose* hardware, seperti Raspberry. Adapun beberapa *Software Switch* yang tersedia antara lain OVS *(Open vSwitch)* dan OpenWRT .

ISBN: 978-602-60581-0-2

2.4 OpenFlow Controller

Untuk mengendalikan jaringan yang diprogram *(programmable network)*, diperlukan sebuah perangkat lunak yang memungkinkan untuk mengirim instruksi untuk dikirim ke perangkat jaringan. Perangkat lunak tersebut diimplementasikan dalam sebuah *controller* yang merupakan otak dari jaringan, yang mampu menjalankan beberapa tugas. OpenFlow controller adalah aplikasi atau perangkat lunak independen yang berjalan di server kusus, dimana bertanggung jawab untuk mengelola OpenFlow *switch. Controller* bertanggung jawab penuh terhadap apa yang terjadi dalam jaringan. *Controller* dapat menambah, menghapus, atau memperbarui *flow table,* dengan menggunakan protokol OpenFlow. *Flow table* merupakan *database* yang menyimpan semua *flow entries* yang terkait dengan *action.,* sehingga *switch* dapat menerapkan tindakan tertentu terhadap *flow* tertentu.

Pox merupakan sebuah *platform* untuk pengembangan dan prototipe dari perangkat lunak kontrol jaringan menggunakan Python [13]. POX juga menyediakan *framework* untuk berkomunikasi dengan SDN *Switch* menggunakan protokol OpenFlow. *Developer* dapat menggunakan POX untuk membuat sebuah controller SDN dengan menggunakan bahasa pemrograman Python. POX juga merupakan sebuah *tool* yang populer yang digunakan sebagai media bagi para *researcher* untuk mempelajari SDN dan *Network Application Programing.* Pada dasarnya, POX dapat digunakan secara langsung sebagai Controller SDN dengan menggunakan *stock component* yang telah di *bundle* dalam POX tersebut.

4. Desain Topology dan Implementasi

4.1 Instalasi Mininet dan Ekstensi Mininet-WiFi pada VM

Untuk menjalankan simulasi SDN yang akan dilakukan, pertama kali dilakukan instalasi Mininet sebagai simulator SDN. Mininet akan diinstal dalam dua bua VM, dimana VM pertama digunakan sebagai *Controller* SDN (POX *Controller*) dan VM kedua adalah Mininet yang ditambahkan ekstensi Mininet-WiFi. Adapun VM tersebut diinstal dalam *Notebook* dengan prosesor *Intel Core i7* 4720HQ clock 2.6 GHz, RAM 8 Gb, dan sistem operasi Windows 10 Pro. VM pertama adalah Mininet yang ditambahkan ekstensi Mininet-WiFi dan VM kedua adalah Mininet yang akan digunakan sebagai controller.

Pada penelitian ini digunakan *image* Mininet versi 2.2.1 yang di unduh pada laman *mininet.org/download.* Karna *image* Mininet yang didownload berbasis Linux Server 14.04 LTS, maka ditambahkan GUI, yaitu *Gnome Desktop.*

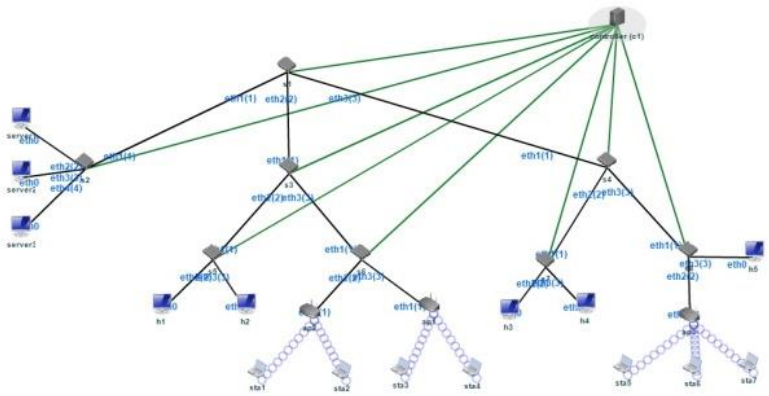
4.2 Desain Topologi

Pada penelitian ini akan digunakan topologi *tree.* Pemilihan jenis topologi ini karena biasa nya *data center* dan *campus* menerapkan jenis topologi ini. Salah satu kelebihan dari topologi ini adalah sangat mudah untuk dikembangkan. Dengan penambahan *switch* dan *Links* akan membangun topologi *tree* yang kompleks *(fat tree).*

Dalam topologi *tree* yang dibangun, terdiri dari beberapa *Open vSwitch, access point* dan *host.* Pada topologi ini, digunakan *access point* supaya *host* yang memiliki antarmuka WiFi *(station)* dapat terhubung dengan *Open vSwitch*. Dalam topologi yang digunakan juga ditambahkan beberapa *performance* parameter dalam *Link* seperti *Bandwidth* (100Mbps), *Delay* (5ms), dan Loss (0%).

Topologi yang digunakan adalah seperti gambar 2 dibawah ini:

ISBN: 978-602-60581-0-2



Gambar 2. Topologi yang digunakan

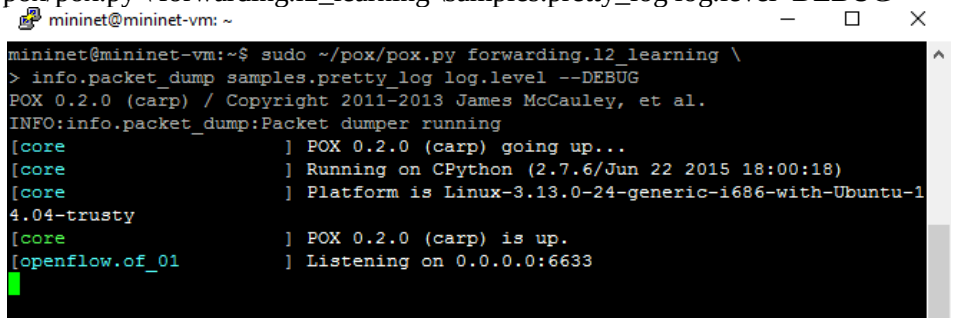
Topologi tersebut terdiri dari; sebuah *controller* (pada VM2) dengan ip 192.168.22.128, delapan buah *Open vSwitch*, tiga buah *access point*, dan 15 *host. Host* dalam topologi terdiri dari 8 *host* yang terhubung dengan *wire* ke *Open vSwitch*, dan 7 *host* atau *station* yang terhubung ke *Open vSwitch* melalui *access point*.

5. Simulasi dan Analisa

5.1 Pengujian Topologi

Setelah topologi terbentuk, maka dilanjutkan dengan men-test topologi tersebut. Sebelum melakukan pengujian, maka *controller* pada VM ke dua harus di aktifkan terlebih dahulu dengan *command:*

Sudo ~/pox/pox.py \ forwarding.l2_learning \samples.pretty_log log.level–DEBUG



Gambar 3 Aktifkan POX controller

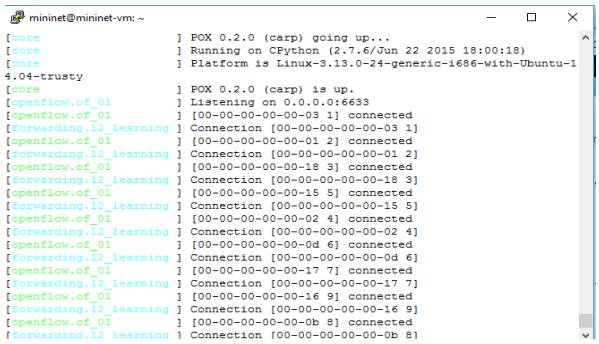
Dalam POX *controller* ini digunakan komponenn *forwarding.l2_learning.* Setelah controller diaktifkan, maka selanjutnya adalah menjalankan topologi melalui terminal linux dengan command:

```
Python topologi_linkBW.py
```

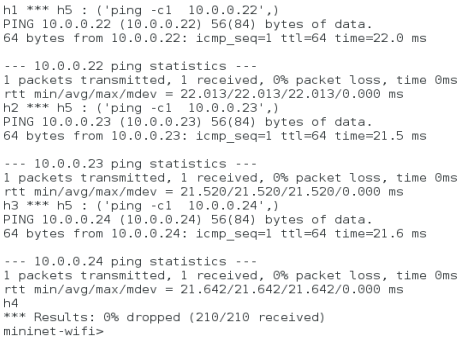
1) *Ping test*

Setelah topologi dijalankan, topologi akan terhubung ke controller, seperti gambar 6 dibawah ini, dan untuk menguji dan memastikan apakah semua *link* telah terhubung dilakkan *pingall*.

ISBN: 978-602-60581-0-2



Gambar 4. POX l2_learning switch terhubung dengan topologi



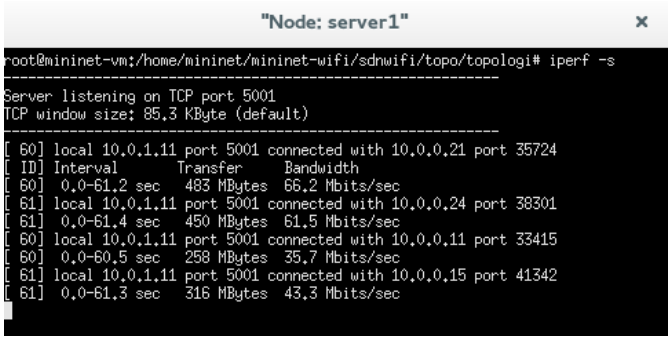
Gambar 5. Tes konektifitas *pingall*

Pada topologi ini, *Open vSwitch* terhubung dengan *Open vSwitch* yang lain denga *link* 100Mbps, dan *Open vSwitch* terhubung dengan host dan *access point* dengan *link* 100Mbps. *Access point* terhubung ke *host/station* dengan kecepatan 54Mbps. Dari gambar 7 diatas dapat dilihat bahwa semua host telah terhubung semua, ditandai dengan tidak adanya paket yang hilang.

2) *Bandwidth Test* dengan *Iperf*

Tes ini bertujuan untuk mengukur *bandwidth* dari topologi yang dibuat dengan menggunakan *iperf tool.* Sebelum melakukan pengukuran, langkah pertama yang harus dilakukan adalah dengan membuat *iperf server* terlebih dahulu. Setelah *iperf server* dibuat, maka pengukuran *bandwidth* antara *client* dengan *server* dapat dilakukan. Pada topologi ini akan dilakukan pengukuran *bandwidth* antara *host* yang terhubung secara *wire* ke *Open vSwitch* dan *host/station* yang terhubung secara *wireless* melalui *access point*.

Tes pertama adalah untuk melihat *bandwidth* antar h1 (10.0.0.21), h4 (10.0.0.24) dengan *iperf server* (server1 ip=10.0.1.11) yang terhubung dengan *Open vSwitch* secara *wire,* dan pada tes kedua adalah melihat *bandwidth* antara *sta1* (10.0.0.11), *sta5* (10.0.0.15) dengan *iperf server* yang terhubung ke *Open vSwitch* melalui *access point*. Antara *Open vSwitch* dengan *host* dihubungkan oleh *Link* dengan *bandwidth* 100Mbps, *delay 5ms*, dan *loss 0. Open vSwitch* juga dihubungkan dengan *access point* oleh oleh Link dengan *bandwidth* 100Mbps, *delay 5ms*, dan *loss 0. Access point* yang digunakan mempunyai *bandwidth* 54Mbps. Pengukuran *throughput bandwidth* dilakukan selama 60 detik, dan hasilnya dapat dilihat pada gambar 6 dibawah ini:



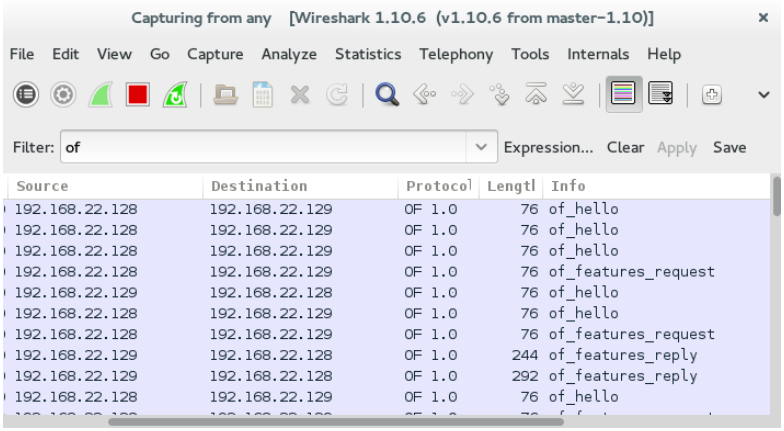
Gambar 6 pengukuran *bandwidth*

Dari gambar 6 diatas, dapat dilihat bahwa bandwidth maksimum antara *h1* dan *server1* adalah 66.2 Mbps, *h4* dengan *server1* adalah 61.5 Mbps, *sta1* dengan *server1* adalah 35.7 Mps, dan bandwidth maksimum antara *sta5* dan *server1* adalah 44,5 Mbps. Dari hasil tersebut dapat dilihat bahwa *controller POX* dapat meng-handle topologi jaringan yang didalamnya terdapat *host* yang terhubung dengan *Open vSwitch* melalui *access point*.

ISBN: 978-602-60581-0-2

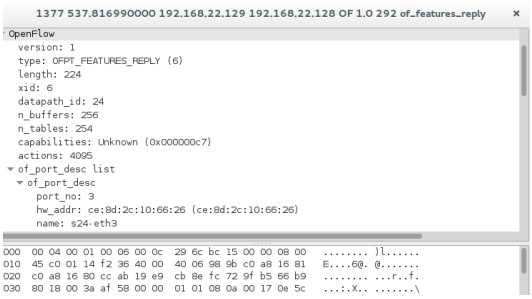
3) Analisa paket OpenFlow dengan *Wireshark*

Ketika melakukan simulasi SDN dengan Mininet, untuk menangkap pesan OpenFlow adalah dengan menggunakan *Wireshark*. Untuk mengidentifikasi *switch* mana yang mengirim atau menerima OpenFlow *messages* adalah dengan melihat sumber atau tujuan dari TCP *port* paket OpenFlow. Hal ini karena sebagian besar OpenFlow yang dipertukarkan antara *switch* dengan *controller* tidak mengandung informasi lainnya yang membantu mengidentifikasi *switch* yang bertindak sebagai sumber atau tujuan dari pesan OpenFlow tersebut. Untuk menangkap pesan OpenFlow, terlebih dahulu kita harus menjalankan *wireshark*, dan memfilter protokol OpenFlow. Topologi dijalankan terlebih dahulu, kemudian baru controller supaya pesan OpenFlow bisa ter-*caputure* di *wireshark.* Adapun hasil capture pesan OpenFlow dapat dilihat pada gambar 7 dibawah ini.

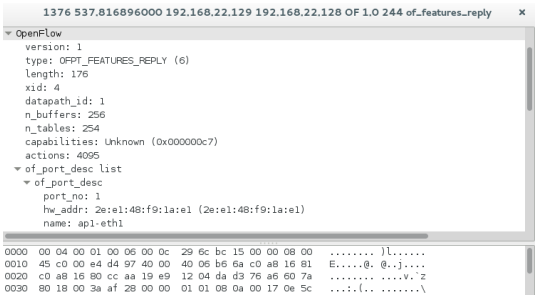


Gambar 7 Pesan OpenFlow dari *Switch* dan *controller*

Dari hasil *capture wireshark* pada gambar 4.7 dapat dilihat bahwa pada waktu pertama kali *controller* diaktifkan akan terdapat pesan *Hello.* Pesan ini digunakan oleh *controller* maupun *Switch* saat pemasangan sambungan *(connection setup)* untuk negosiasi versi OpenFlow. Ketika *controller* dan *Switch* telah tersambung, maka *controller* dan *Switch* akan mengirim pesan *hello.* Jika koneksi gagal, maka pesan yang dikirim adalah *HelloFailed.*



Gambar 8. Pesan OpenFlow dari OVS



Gambar 8. Pesan OpenFlow dari AP

Pada gambar 8a dan 8b juga dapat dilihat bahwa terdapat pesan *feature request.* Pesan ini mengindikasikan bahwa *controller* menanyakan port mana saja yang tersedia dari *Switch.* Sedangkan *feature reply* merupakan pesan yang dikirim oleh *Switch* ke *controller*, yang berisi *list port, action*, dan *speed* dari *port* yang tersedia. POX *controller* meminta *feature request* ketika pertama kali dijalankan dan setiap *Switch* akan memberi respon dengan *feature reply.* Di dalam pesan *feature reply* terdapat *datapath ID*. Pada gambar 8a terdapat *datapath ID:* 24, ini merupakan *datapath ID* untuk *Open vSwitch* nomor 8 pada topologi, dan Pada gambar 8b terdapat *datapath ID:* 1, ini merupakan *datapath ID* untuk

ISBN: 978-602-60581-0-2

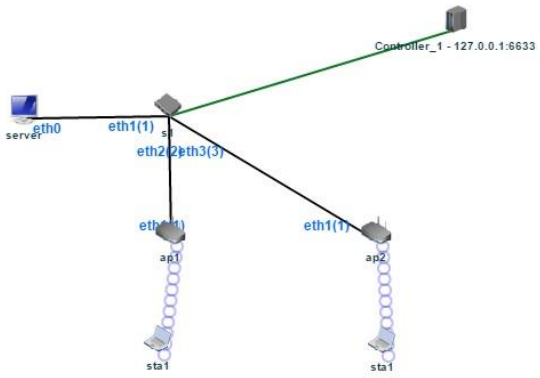
Access Point (AP1) pada topologi. Dapat dilihat, AP mengirim dan menerima pesan OpenFlow dari *controller,* hal ini mengindikasikan bahwa AP bertindak sebagai *Switch.*

5.2 Mobility pada SDN

Pada skenario kali ini akan dibahas mengenai *Mobility* dalam SDN. Adapun *Mobility* dalam jaringan merupakan kemampuan jaringan untuk meng-handle *host* yang bergerak dari suatu titik ke titik yang lain. Setelah penambahan ekstensi Mininet-WiFi, maka sekarang Mininet mampu untuk membuat skenario Mobility, dimana *host* atau *station* yang terhubung dengan suatu *access point* akan bergerak menjauh dari jangkauan *access point* tersebut dan terkoneksi ke *access point* lainnya.

1) Mobility dengan single switch

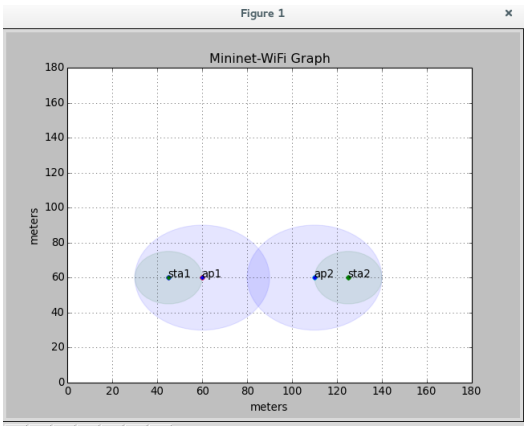
Percobaan pertama digunakan topologi dengan sebuah *Open vSwitch*, dua *access point*, dan dua buah *host (station),* serta sebuah *host* yang digunakan sebagai server. Topologi tersebut dapat dilihat seperti gambar 9 dibawah ini:



Gambar 9. topologi mobility single switch

Pada gambar 9 diatas dapat dilihat bahwa sta1 dan sta2 terhubung ke *Open vSwitch* melalui *access point* yang berbeda. Sta1 terhubung dengan AP1 dan sta2 terhubung dengan AP2. Controller yang digunakan adalah *default* controller pada mininet *(reference controller).*

Dalam skenario ini, sta1 terhubung ke AP1 dan sta2 terhubung ke AP2. Setelah 20 detik, sta1 akan bergerak menjauh dari AP1 dan akan menuju AP2, dan juga sta2 akan bergerak menjauh dari AP2 menuju AP1, dan ketika memasuki kawasan AP yang baru, maka sta1 akan terhubung ke AP2, dan sta2 akan terhubung ke AP1 seperti tampak pada gambar 10a dan 10b. Pemilihan *access point* berdasarkan sinyal terkuat.



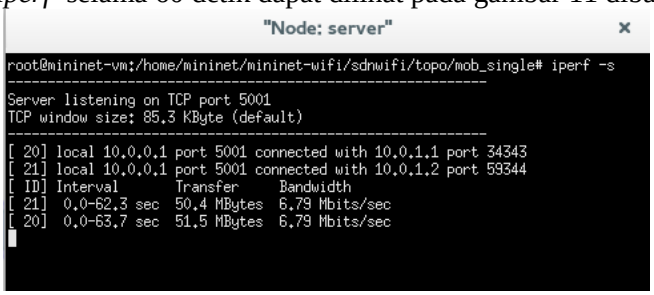
Gambar 10a. Kedaan sebelum *station* bergerak

```
mininet-wifi> info sta1
--------------------------------
Interface: sta1-wlan0
Associated To: ap1
Frequency: 2.412 GHz
Signal level: -39.62 dbm
Tx-Power: 14 dBm
mininet-wifi> info sta2
--------------------------------
Interface: sta2-wlan0
Associated To: ap2
Frequency: 2.412 GHz
Signal level: -39.62 dbm
Tx-Power: 14 dBm
mininet-wifi>
```

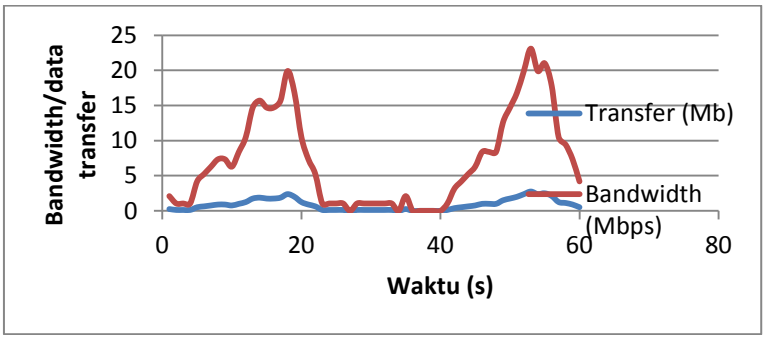
Gambar 10b, interkoneksi *host/sta* ke AP

ISBN: 978-602-60581-0-2

Setelah 20 detik, masing-masing *station* akan bergerak, sta1 akan menuju AP2 dan sta2 akan menuju AP1. Untuk melihat apakah *station* dapat terkoneksi dengan *access point* yang baru adalah dengan mengukur *bandwidth* dan juga untuk mengetes simulasi ini. Mmasing-masing station sebelum bergerak telah dihubungkan dulu ke sebuah *server* (ip=10.0.0.1) yang juga bertindak sebagai *iperf server*. Dalam skenario ini akan diamati proses perpindahan dan *bandwidth* yang terukur ketika masing-masing *station* diam dan mulai bergerak menjauhi AP yang terkoneksi saat ini menuju *access point* yang lain. Jarak antara sta1 dengan AP1 sama dengan jarak AP2 ke sta2 yaitu 15 meter, dan jarak antara AP1 dan AP2 adalah 50 meter. Hasil pengukuran dengan *iperf* selama 60 detik dapat dilihat pada gambar 11 dibawah ini:



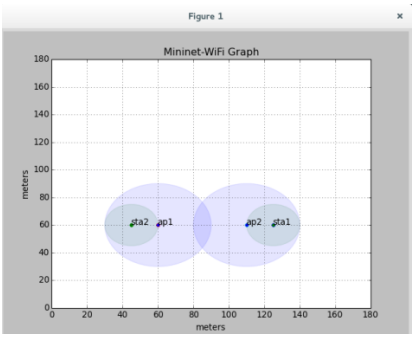
Gambar 11. Iperf test pada skenario Mobility dengan topologi single switch



Gambar 12. *Iperf* dari sta1 ke *server* ketika bergerak dan menemukan AP baru

Dari gambar 11 dapat dilihat, ketika sta1 dan sta2 belum bergerak (jarak sta1 ke AP1 adalah 15 meter, dan juga jarak sta2 ke AP2 adalah 15 meter), bandwidth yang diperoleh sekitar 2 Mbps (gambar 12). Pada gamabr 12 juga dapat dilihat, ketika *station* mulai bererak dan mendekati *access point*, *bandwidth* yang terukur semakin meningkat dan ketika menjauh dari AP *bandwidth* yang terukur akan menurun. Ketika *station* menemukan *access point* baru dengan sinyal yang lebih kuat daripada *access point* sebelumnya, maka station akan terhubung ke *access point* baru. Dari gambar 11 tersebut juga dapat dilihat bahwa ketika bandwidth yang terukur 0Mbps (pada detik 36-40), sta1 tidak terhubung ke *access point* yang lama (AP1) dan sedang mencoba untuk menyambung ke *access point* yang baru. Setelah masing-masing station terkoneksi dengan *access point* yang baru, dan berjalan mendekati *access point*, *bandwidth* yang terukur semakin besar. Sta1 membutuhkan waktu sekitar 5 detik untuk terhubung ke AP2, dan sta2 membutuhkan waktu sekitar 6 detik untuk terhubung ke AP1. Dengan kembali terhubungnya *station/host* dengan *access point* yang baru, maka proses mobility berhasil. Interkoneksi antara *station* dengan *access point* yang baru dapat dilihat pada gambar 13a dan 13b dibawah ini:

ISBN: 978-602-60581-0-2



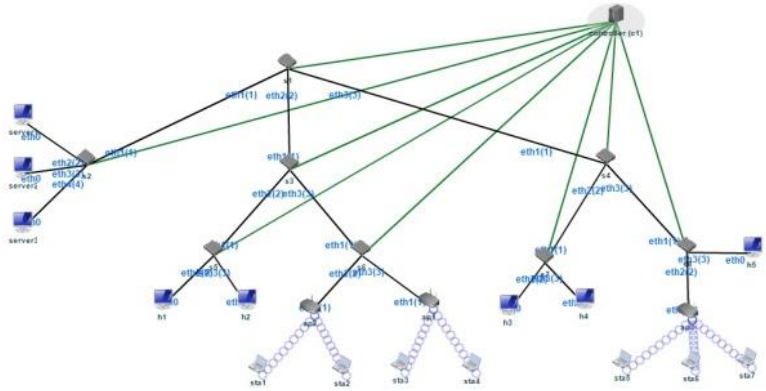
Gambar 13a. *host/sta* terhubung ke AP baru

```
mininet-wifi> info sta1
--------------------------------
Interface: sta1-wlan0
Associated To: ap2
Frequency: 2.412 GHz
Signal level: -39.62 dbm
Tx-Power: 14 dBm
mininet-wifi> info sta2
--------------------------------
Interface: sta2-wlan0
Associated To: ap1
Frequency: 2.412 GHz
Signal level: -39.62 dbm
Tx-Power: 14 dBm
mininet-wifi>
```

Gambar 13b. interkoneksi *host/sta* ke AP baru

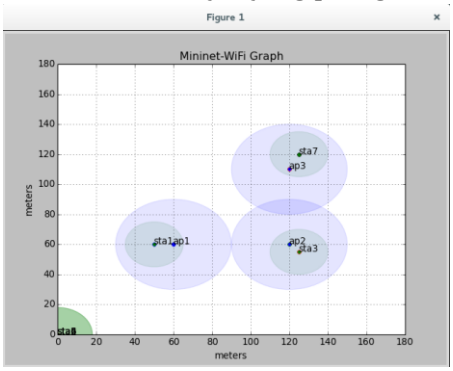
2) Topologi Tree

Skenario kedua adalah menggunakan topologi yang sama seperti Gambar 14. Dalam topologi tree ini terdapat beberapa buah *host/station* dan beberapa *Open vSwitch*. Dalam skenario ini digunakan POX controller dengan komponen *l2_learningswitch*. Penggunaan POX dan komponen ini adalah karena dalam topologi digunakan beberapa *Open vSwitch*. Dalam skenario ini juga dilihat apakah POX controller mampu men-handle *Mobility* ketika *host/station* yang terhubung dengan OVS melalui *access point*.



Gambar 14 Topologi Tree

Dalam skenario ini digunakan 3 buah *access point* dan 3 buah *station.* Interkoneksi antara *station* dengan AP dapat dilihat pada gambar 14a dan 14b dibawah ini. Pemilihan *access point* yang baru berdasarkan sinyal yang paling kuat.




```
--------------------------------
Interface: sta1-wlan0
Associated To: ap1
Frequency: 2.412 GHz
Signal level: -36.10 dbm
Tx-Power: 14 dBm
mininet-wifi> info sta3
--------------------------------
Interface: sta3-wlan0
Associated To: ap2
Frequency: 2.412 GHz
Signal level: -33.09 dbm
Tx-Power: 14 dBm
mininet-wifi> info sta7
--------------------------------
Interface: sta7-wlan0
Associated To: ap3
Frequency: 2.412 GHz
Signal level: -37.06 dbm
Tx-Power: 14 dBm
mininet-wifi>
```

Gambar 15 Interkoneksi *station* dan AP sebelum bergerak

Dalam skenario yang dibuat, masing-masing *station* akan terkoneksi seperti gambar 14 diatas, dan setelah 40 detik, masing-masing *station* akan bergerak, *sta3* akan menuju AP1, *sta1* dan *sta7* akan menuju AP2, dan semua *station* ini akan terkoneksi ke *access point* yang baru. Untuk mengukur *bandwidth* dan juga untuk mengetes simulasi mobility ini, maka masing-

ISBN: 978-602-60581-0-2

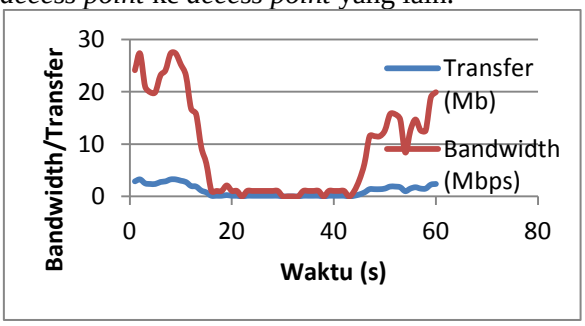
masing station sebelum bergerak telah dihubungkan dulu ke sebuah *server1* (ip=10.0.1.11) yang juga bertindak sebagai *iperf server*. Dalam skenario ini akan diamati proses perpindahan dan *bandwidth* yang terukur ketika masing-masing *station* diam dan mulai bergerak menjauhi *access point* yang terkoneksi saat ini menuju *access point* yang lain. Hasil pengukuran dengan *iperf* dapat dilihat pada gambar 15 dibawah ini:

1. *Iperf test*

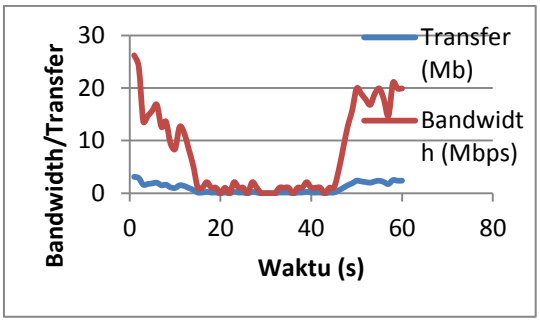
Untuk mengukur *bandwidth* dan juga untuk mengetes simulasi ini, maka masing-masing station sebelum bergerak telah dihubungkan dulu ke sebuah *server* yang juga bertindak sebagai *iperf server*. Dalam skenario ini akan diamati proses perpindahan dan *bandwidth* yang terukur ketika masing-masing *station* diam dan mulai bergerak menjauhi *access point* yang terkoneksi saat ini menuju *access point* yang lain. Adapun hasil pengukuran yang dilakukan adalah *throughput* dari (ketika skenario *mobility* terjadi):

Sta1 (10.0.0.11) ke *server1* (10.0.1.11), *throughput bandwidth* 8.95 Mbps
Sta2 (10.0.0.12) ke *server2* (10.0.1.12), *throughput bandwidth* 7.78 Mbps
Sta3 (10.0.0.13) ke *server3* (10.0.1.13), *throughput bandwidth* 9.09 Mbps

Tujuan pengukuran adaah untuk melihat throughput *bandwidth* ketika *station* bergerak dan untuk melihat apakah *controller* POX mampu menangani proses perpindahan *station* dari suatu *access point* ke *access point* yang lain.



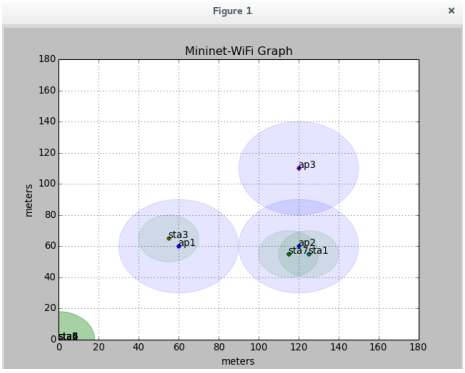
Gambar 16a *iperf* sta1 ke server1



Gambar 16b *iperf* sta2 ke server2

Dari gambar 16a dan 16b dapat dilihat, sebelum masing-masing *station* mulai, bandwidth yang diperoleh sekitar 25 Mbps. Ketika bergerak menjauh dari *access point* yang saat ini terkoneksi, *bandwidth* yang terukur semakin lama akan semakin menurun. Ketika *station* menemukan *access point* baru dengan sinyal yang lebih kuat daripada AP sebelumnya, maka *station* akan terhubung ke *access point* baru. Dari gambar 16 tersebut juga dapat dilihat bahwa ketika *bandwidth* yang terukur 0Mbps (sta1 detik 30-33, sta2 detik 29-32), *station* tersebut tidak terhubung ke *access point* manapun dan sedang mencoba untuk terhubung dengan *access point* baru. Sta1 membutuhkan waktu sekitar 4 detik untuk terhubung ke AP2, sta3 membutuhkan waktu sekitar 4 detik untuk terhubung ke AP1, dan sta7 membutuhkan waktu sekitar 4 detik untuk terhubung ke AP1. Dari hasil ini juga dapat dilihat bahwa POX controller mampu untuk menangani proses *Mobility* pada SDN, ditandai dengan terhubung kembalinya *station* dengan v yang baru dan semakin lama *bandwidth* yang terukur semakin besar seiring dengan mendekatnya jarak antara *access point* denga *station/host* tersebut.

ISBN: 978-602-60581-0-2



Gambar 17a. *host/sta* terhubung ke *access point* baru

```
Interface: sta1-wlan0
Associated To: ap2
Frequency: 2.412 GHz
Signal level: -33.08 dbm
Tx-Power: 14 dBm
mininet-wifi> info sta3
--------------------------------
Interface: sta3-wlan0
Associated To: ap1
Frequency: 2.412 GHz
Signal level: -33.09 dbm
Tx-Power: 14 dBm
mininet-wifi> info sta7
--------------------------------
Interface: sta7-wlan0
Associated To: ap2
Frequency: 2.412 GHz
Signal level: -33.09 dbm
Tx-Power: 14 dBm
mininet-wifi>
```

Gambar 17b. interkoneksi *host/sta* ke *access point* baru

6. Kesimpulan

Saat ini SDWN merupakan paradigma baru dalam jaringan nirkabel, yang secara fisik memisahkan control plane dan data plane dari berbagai macam elemen infrastruktur wireless. SDWN ini merupakan cabang dari SDN untuk infrastruktur *wireless*. Dengan menambahkan ekstensi Mininet-WiFi pada Mininet, *researcher* dapat menambahkan *access point* sehingga *host* bisa dihubungkan ke *Open vSwitch* melalui *access point* tersebut. Hal ini memberikan manfaat dalam hal memfasilitasi dan memvalidasi percobaan SDWN.

Pada skenario *mobility*, *host* yang terhubung secara *wireless* ke AP dapat terhubung ke *access point* yang baru ketika host tersebut bergerak dan menemukan AP yang baru, yang mana pemilihan *access point* berdasarkan kekuatan sinyal. Pada saat menangkap pesan OpenFlow dengan *wireshark, access point* mengirim dan menerima pesan OpenFlow dari *controller*, hal ini mengindikasikan bahwa *access point* bertindak sebagai *Switch.* Dalam skenario *mobility* menggunakan topologi *single Switch* (menggunakan *reference controlle)* berhasil dilakukan proses *mobility*, dimana masing-masing *station* berhasil terhubung ke *access point* yang baru ditandai dengan naiknya *bandwidth* ketika *station* tersebut terhubung ke *access point* yang baru setelah sebelumnya sempat putus sekitar 5 detik. Pada skenario mobility dengan topologi *Tree* dengan menggunakan *controller* POX juga berhasil dilakukan skenario Mobiity. Masing-masing *station* berhasil terhubung kembal ke *access point* yang baru setelah sebelumnya berada diluar jangkauan *access point* yang lama ketika proses mobility berlangsung, yang sebelumnya sempat terputus selama 4 detik sebelum terhubung kembali ke *access point* yang baru. Dari penelitian ini dapat dilihat bahwa *controller* POX dapat meng-handle proses Mobility.

Daftar Pustaka

[1] Open Networking Foundation. www.opennetworking.org. [Online]. [cited 2016 September 10. Available from: https://www.opennetworking.org/sdn-resources/sdn-definition.

[2] Nachikethas JA, Krishnamachari B. Software-Defined Networking Paradigms in Wireless Networks: A Survey. ACM Computing Surveys (CSUR). 2015 January; 47(2): p. 1-11.

[3] Sama MR, Contreras LM, Kaippallimalil J, Akiyoshi I. Software-defined control of the virtualized mobile packet core. IEEE Communications Magazine. 2015; 53(2): p. 107-115.

[4] Costanzo S, Galluccio L, Morabito G, Palazzo S. Software Defined Wireless Networks: Unbridling SDNs. European Workshop on Software Defined Networking (EWSDN). 2012 October.

[5] Costanzo S, Gallucio L, Morabito G, Palazzo S. Software Defined Wireless Network (SDWN): an evolvable architecture for W-PANs. 2015 IEEE 1st International Forum on

ISBN: 978-602-60581-0-2

Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI). 2015 September.

[6] He D, Chan S, Guizani M. Securing software defined wireless networks. IEEE Communications Magazine. 2016; 54(1): p. 20-25.

[7] Wang W, Chen Y, Zhang Q, Jiang T. A Software-Defned Wireless Networking Enabled Spectrum Management Architecture. IEEE Communications Magazine. 2016 January; 54: p. 33-39.

[8] Lantz B, Heller B, Mckeown N. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. Proceeding Hotnets-IX Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. 2010;: p. 19:1-19:6.

[9] Duan Q, Yan Y, Vasilakos AV. A Survey on Service-Oriented Network Virtualization Toward Convergence of Networking and Cloud Computing. IEEE Transactions on Network and Service Management. 2012 December; 9(4): p. 373-392.

[10] Sezer S, Hayward SS, Chouhan PK, Fraser B, Lake D, Finnegan J, et al. Are we ready for SDN? Implementation challenges for software-defined networks. IEEE Communications Magazine. 2013 July; 51(7): p. 36-43.

[11] Mulyana E. Buku Komunitas SDN-RG Mulyana E, editor. Bandung: GitBook; 2014.

[12] Open Networking Foundation. OpenFlow Switch Specification Version 1.5.0 (Protocol version 0x06). 15th ed. Foundation ON, editor.: Open Networking Foundation; 2014.

[13] McCauley M. openflow.stanford.edu. [Online].; 2015 [cited 2016 October 27. Available from: https://openflow.stanford.edu/display/ONL/POX+Wiki.

ISBN: 978-602-60581-0-2