Simulasi Konsep *Software Defined Network* (SDN) Menggunakan Raspberry Pi

Teguh Indra Bayu[1], Etvan Ewaldo Tahan[2]
[1,2]*Teknik Informatika,Universitas Kristen Satya Wacana*
Jl. Dr. O. Notohamidjojo, Salatiga 50714, Indonesia
teguh.bayu@uksw.edu
672014045@student.uksw.edu

Abstract*— There are many ways to manage a computer network such as bandwidth management, file permission, etc. However, the technicians are still configuring one by one to each switch and router device in a network infrastructure. Then the idea emerged from the researchers in the form of a concept of Software Defined Network (SDN), which network management would center on one controller and switch openflow. Access Control List (ACL) is an alternative to secure a computer network. By applying Software Defined Network (SDN) concept which is based on management concept of centralized network use software. It focus on how applying ACL in infrastructure network SDN use Raspberry Pi as an openflow switch. The decisive element of an infrastructure network SDN is a controller, an openflow switch, and network device would connect one network to the other network by using Openflow protocol. From the research which is conducted SDN concept, ACL able to applied using flow-control method. It gives some clear value on flow-control, so the restriction access can be done in every network device which is connected in infrastructure network SDN.

Intisari— Banyak cara dalam memanajemen suatu jaringan komputer contohnya seperti manajemen *bandwidth*, manajemen hak akses, dan sebagainya. Namun, para teknisi masih melakukan konfigurasi secara satu persatu kepada setiap perangkat switch dan router di dalam sebuah infrastruktur jaringan. Kemudian muncul ide dari para peneliti yang berupa sebuah konsep *Software Defined Network* (SDN), yang dimana manajemen jaringan akan berpusat kepada satu *controller* dan *switch openflow*. *Access Control List* (ACL) merupakan sebuah alternatif untuk mengamankan sebuah jaringan komputer. Dengan menerapkan konsep SDN yang didasarkan pada sebuah konsep manajemen jaringan terpusat menggunakan perangkat lunak. Simulasi ini berfokus pada bagaimana menerapkan ACL pada infrastruktur jaringan SDN menggunakan Raspberry Pi sebagai *openflow switch*. Unsur penentu dari terbentuknya sebuah infrastruktur jaringan SDN adalah *controller, openflow switch*, dan perangkat jaringan yang akan terhubung satu dengan lainnya menggunakan protokol Openflow. Dari hasil penelitian yang dilakukan dengan konsep SDN, ACL mampu diterapkan dengan menggunakan metode *flow-control*. Dengan memberikan nilai-nilai parameter yang jelas pada *flow-control*, maka pembatasan hak akses dapat dilakukan pada setiap perangkat jaringan yang terhubung di dalam infrastruktur jaringan SDN.

***Kata Kunci*— ACL, SDN, Raspberry Pi, Openflow**

I. PENDAHULUAN

Banyak cara dalam memanajemen suatu jaringan komputer contohnya seperti manajemen *bandwidth*, manajemen hak akses, dan sebagainya. Namun, para teknisi masih melakukan konfigurasi secara satu persatu kepada setiap perangkat *switch* dan *router* di dalam sebuah infrastruktur jaringan. Kemudian muncul ide dari para peneliti yang berupa sebuah konsep yang bernama *Software Defined Network* (SDN). Banyak pula penelitian yang dilakukan untuk meningkatkan manajemen jaringan demi mendapatkan kinerja aliran yang lebih baik. Pada literatur [1], *Software Defined Network* (SDN) digunakan untuk membantu memanajemen sebuah infrastruktur jaringan. Namun, dalam penelitian tersebut tidak lakukan pengujian langsung pada perangkat *switch* dikarenakan mereka menggunakan Mininet sebagai *switch virtual* sehingga mereka tidak perlu menggunakan perangkat *switch* yang sebenarnya. Dalam penelitian yang akan dilakukan ini, penulis akan secara langsung menerapkan konsep SDN ke dalam perangkat keras jaringan dengan menggunakan fungsi *Access Control List* (ACL). ACL merupakan sebuah metode pemberian hak akses dalam sebuah jaringan komputer berdasarkan *MAC Address/IP Address*. ACL dapat menjadi sebuah alternatif untuk mengamankan sebuah jaringan komputer, karena ACL mampu membatasi *traffic* jaringan, memberikan dasar keamanan untuk akses ke jaringan, memberi keputusan terhadap jenis *traffic* mana yang akan dilewatkan atau di-*block* melalui *interface router*, mengontrol daerah-daerah dimana *client* dapat mengakses jaringan, dan menentukan *host* yang diijinkan atau di-*block* akses ke segmen jaringan [2].

SDN merupakan konsep baru dalam dunia jaringan yang memanfaatkan bahasa pemrograman tertentu untuk mengatur/memanajemen sebuah jaringan komputer, yang dimana kontrol jaringan menjadi terpusat. Konsep SDN memerlukan sebuah perangkat yang mampu menjadi pengontrol jaringan yang disebut dengan *controller*. Di dalam *controller* tersebut, kita mampu menerapkan berbagai fungsi pengontrolan jaringan, termasuk fungsi ACL itu sendiri.

Rumusan masalah pada penelitian ini adalah bagaimana merancang dan menerapkan fungsi ACL pada konsep SDN

dengan menggunakan *controller* Opendaylight dan *openflow* sebagai protokol yang digunakan. Berdasarkan latar belakang yang telah dijelaskan, maka penelitian ini bertujuan untuk mengetahui bagaimana menerapkan fungsi ACL pada konsep SDN. Manfaat dari penelitian ini adalah guna membuat pembaca mengerti bagaimana menerapkan fungsi ACL ke dalam *controller* Opendaylight dalam konsep SDN.

II. LANDASAN TEORI

Pada penelitan yang berjudul *Software Defined Wireless Network Testbed using Raspberry Pi OF Switches with Routing Add-on*, disimpulkan bahwa penelitian tersebut bertujuan untuk mengimplementasikan konsep SDN untuk infrastruktur jaringan *wireless* (nirkabel) dengan menggunakan Raspberry Pi sebagai Openflow (OF) *switch*. Selain untuk pengimplementasian konsep SDN, penelitian tersebut juga bertujuan untuk meningkatkan pemanfaatan jaringan dengan mengatur perbedaan jenis *traffic* dari banyak *user* dalam infrastruktur jaringan tersebut [3]. OpenWRT [4] digunakan menjadi sistem operasi yang digunakan oleh *openflow switch* dalam penelitian tersebut. Dengan sistem operasi berbasis linux, OpenvSwitch diinstall agar Raspberry Pi dapat digunakan sebagai *openflow switch*. Perbedaan dari penelitian yang akan dilakukan adalah jaringan yang digunakan berbasis *Local Area Network* (LAN) sehingga pertukaran data yang dilakukan akan lebih stabil.

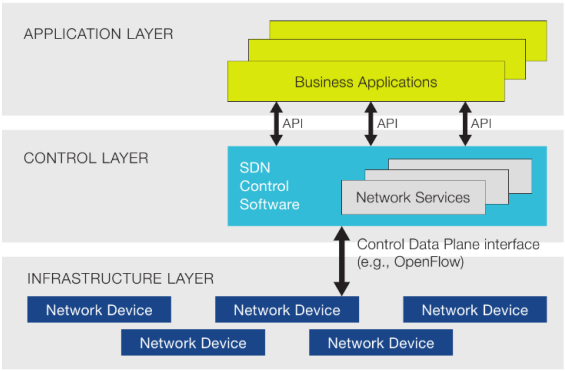
Pada penelitian lainnya yang berjudul *Software Defined Networking (SDN)*, dalam pengujian *bandwith testing* pada Raspberry Pi yang diinstall OpenvSwitch sebagai *openflow switch* dan Mikrotik yang diinstall OpenWrt yang juga digunakan sebagai *openflow switch* disimpulkan bahwa dalam perihal *limit banwidth* pada Raspberry Pi belum cukup kuat menandingi kecepatan transfer Mikrotik pada protokol TCP dan UDP antar *client* dalam infrastruktur jaringan SDN dalam penelitian tersebut [5]. Perbedaan dari penelitian yang akan dilakukan adalah *openflow switch* akan menggunakan Raspberry Pi sebagai *switch*-nya dengan impelementasi aplikasi OpenvSwitch pada Raspberry Pi.

Pada penelitian terdahulu selanjutnya yang berjudul *The Integration of Access Control Levels Based on SDN*, penelitian ini bertujuan untuk menganalisis ACL di dalam konsep SDN dengan tabel kontrol yang dinamis sehingga *rules* yang diterapkan dapat berubah berdasarkan aktivitas jaringan dan disimpulkan bahwa tabel kontrol dinamis dapat diterapkan untuk konsep SDN [6]. Dalam penelitian yang akan dilakukan, implementasi ACL pada konsep SDN akan dilakukan sama seperti penelitian sebelumnya, namun tabel kontrol tidak dinamis melainkan statis atau kontrol akses tidak berubah-ubah.

Berdasarkan penelitian terdahulu terkait SDN, maka akan dilakukan penelitian tentang simulasi konsep Software Defined Network (SDN) menggunakan Raspberry Pi. Tujuannya mengimplementasi fungsi ACL pada *controller* Opendaylight yang terhubung dengan Raspberry Pi sebagai *openflow switch*, sehingga dapat diketahui cara merancang dan menerapkan fungsi ACL pada konsep SDN. Dalam pengujiannya, akan diterapkan fungsi ACL ke dalam *controller* yang akan terhubung pada *server* dan *client*.

A. *Software Defined Network (SDN)*

Software Defined Network (SDN) adalah sebuah paradigma baru dalam mendesain, mengelola, dan mengimplementasikan jaringan, terutama untuk mendukung kebutuhan inovasi dibidang jaringan yang semakin lama semakin kompleks. Konsep dasar SDN adalah melakukan pemisahan antara *control* dan *forwarding plane/data plane*. Dengan menggunakan konsep SDN, sebuah infrastruktur dapat dikelola/dimanajemen dengan mudah, dikarenakan adanya *controller* yang mengatur setiap *router* dengan protokol Openflow [7]. SDN memiliki tanggung jawab dalam mengatur *control plane*, mulai dari mendeteksi perangkat jaringan yang akan diteruskan sampai kepada memanajemen perangkat jaringan secara terpusat dengan menggunakan *software* melalui abstraksi dan pandangan yang luas kepada seluruh jaringan [8].



Gambar. 1. *Basic SDN Architecture* [9]

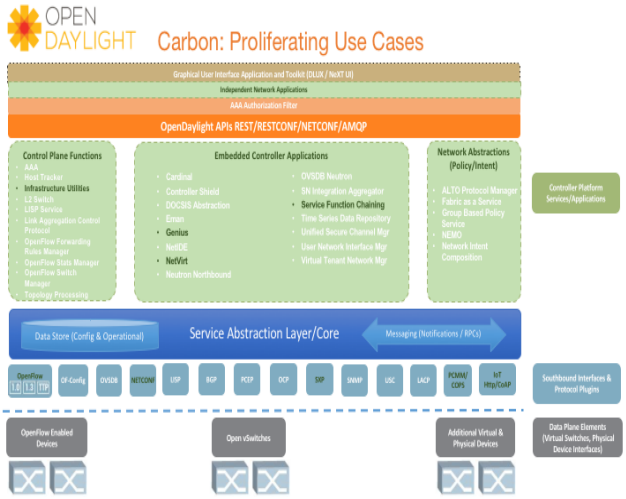
Berdasarkan Gambar 1, konsep SDN menggunakan *Application Programming Interface* (API) untuk menghubungkan *application layer* dan *control layer*, dimana diperlukan bahasa pemrograman tertentu untuk mengkonfigurasi *controller* yang nantinya akan terhubung ke infrastruktur jaringan yang dibuat. Sementara, *control layer* dan *infrastructure layer* dihubungkan oleh protokol *openflow*. SDN juga mendukung pengguna untuk dapat mengembangkan aplikasi pengontrol jaringan. *Control plane* berguna untuk mengontrol jaringan, sedangkan *data plane* berfungsi untuk meneruskan sebuah *packet* dan menentukan bagaimana "reaksi" atas *packet* yang diterima atau dilewatkan. Tujuan dari SDN adalah membantu mempermudah pengontrolan sebuah infrastruktur jaringan.

B. *Opendaylight*

Opendaylight merupakan sebuah proyek *open source* yang digunakan sebagai pengontrol jaringan pada konsep SDN. *Controller* ini diimplementasikan pada perangkat lunak yang ditampung dalam *Java Virtual Machine* (JVM). Opendaylight dapat digunakan pada *hardware* dan *platform* sistem operasi apapun yang mendukung Java [10]. Aplikasi Opendaylight menggunakan *controller* untuk mengumpulkan data-data jaringan, menjalankan algoritma analisis dalam infrastrukur jaringan, lalu menerapkan aturan

JUTEI Edisi Volume.2 No.2 Oktober 2018
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.2018.22.100

baru ke seluruh jaringan. Bahasa pemrograman yang digunakan adalah python dan C++. Berikut adalah gambar bagan dari Opendaylight.

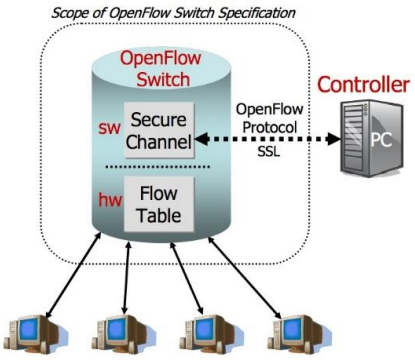


Gambar. 2. Opendaylight *Architecture for Carbon Release* [11]

Pada Gambar 2, merupakan struktur dari *software* Opendaylight, dapat dilihat banyak layanan yang dapat digunakan dalam manajemen sebuah infrastrukur jaringan. Opendaylight menggunakan RESTCONF API untuk berhubungan dengan *application layer*. RESCONT adalah sebuah REST protokol yang berjalan di bawah *interface* HTTP guna dapat mengakses *server* NETCONF *datastore*, di dalam *datastore* terdapat data-data perangkat yang terhubung ke *controller* Opendaylight. Kemudian untuk *controller* Opendaylight dapat berhubungan dengan perangkat jaringan digunakan protokol *openflow*, sebagai protokol penyokong dari infrastruktur jaringan SDN.

C. *Openflow*

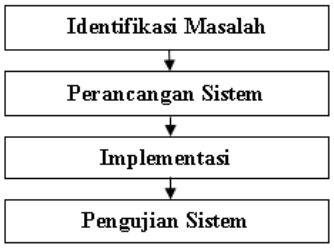
Openflow merupakan standar dari konsep SDN. Tugas dari *openflow* adalah menjadi protokol penghubung antara *controller* Opendaylight dengan infrastruktur jaringan. Cara kerja dari protokol *openflow* yaitu memungkinkan akses langsung dan memanipulasi *forwarding plane* pada perangkat jaringan seperti *switch* dan *router*, baik fisik maupun virtual, sehingga nantinya terhubung ke *controller* dan dapat lebih beradaptasi dengan kebutuhan. Dalam lingkungan *openflow*, perangkat apapun yang ingin berkomunikasi dengan *controller* SDN harus didukung dengan protokol *openflow*. Melalui *interface* ini, *controller* SDN dapat melakukan perubahan pada *switch/router flow-table* yang memungkinkan administrator jaringan untuk memanajemen *traffic*, mengontrol aliran data untuk kinerja yang optimal, dan mulai menguji konfigurasi dan aplikasi baru *[9]*. Pada Gambar 7 ditampilkan kerangka dari standar arsitektur *openflow*.



Gambar. 3. Openflow Architecture [12]

III. METODOLOGI PENELITIAN

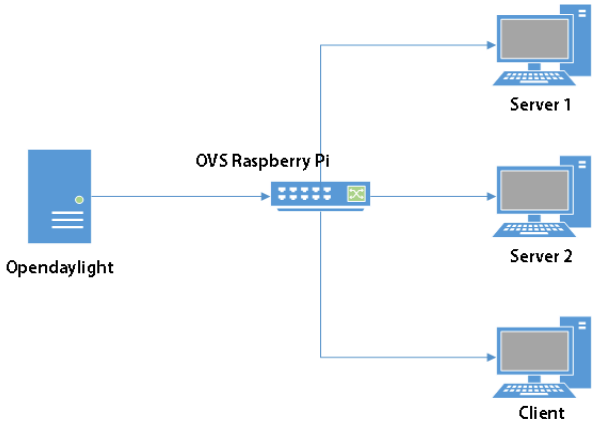
Tahapan penelitian yang digunakan dalam membuat Simulasi Konsep *Software Defined Network* (SDN) menggunakan Raspberry Pi, dapat dilihat pada Gambar 4.



Gambar. 4. Tahapan Penelitian

Tahap-tahap dalam tahapan penelitian yang ada pada Gambar 4, dijelaskan sebagai berikut: Pada tahap Identifikasi Masalah: Dalam lingkungan kerja seperti kantor, pastinya sangat diperlukan jaringan komputer untuk membantu pegawai/*user* untuk dapat bertukar data satu sama lain. Permasalahan jaringan yang sering dijumpai adalah ketika ada *user* dapat mengakses setiap *server* sehingga dapat mengganggu kinerja dari *user* lain. Karena dalam lingkungan kerja banyak pegawai yang menggunakan jaringan komputer, setiap pegawai mempunyai tugas masing-masing maka dari itu kita perlu memberikan hak akses masing-masing kepada *user* untuk membatasi kemana *user* tersebut dapat mengakses data dan bergantung pada tugas yang ada. Untuk itu, kita dapat menerapkan ACL pada infrastruktur jaringan yang ada. Dalam penelitian ini infrastrukur yang digunakan adalah *Software Defined Network* (SDN), dimana kontrol terhadap jaringan berbasis aplikasi dan terpusat dengan menggunakan protokol *openflow*.

Perancangan Sistem: Pada tahap ini akan dilakukan analisis permasalahan dan kebutuhan dalam perancangan yang berkaitan dengan simulasi penerapan ACL pada konsep SDN. Perancangan ini dimulai dari skema topologi jaringan sederhana yang akan diterapkan dalam pembuatan infrastruktur SDN yang akan diberikan ACL di dalamnya. Kemudian dilanjutkan dengan perancangan fungsi ACL yang akan diterapkan ke dalam *swith* melalui *controller*. Topologi jaringan SDN dapat dilihat pada Gambar 5



Gambar. 5. Topologi Jaringan

Dapat dilihat pada Gambar 5, *controller* Opendaylight terhubung dengan Raspberry Pi, di dalamnya akan diinstall OpenvSwitch sehingga Raspberry Pi dapat digunakan sebagai *switch*. OpenvSwitch merupakan sebuah *software* yang dapat membuat PC virtual maupun fisik mempunyai kinerja yang sama dengan *switch*. Setelah OpenvSwitch diinstall, akan dilakukan konfigurasi protokol Openflow pada setiap interface yang ada pada *switch* dan kemudian diperlukan modul LAN tambahan karena pada Raspberry Pi hanya tersedia satu *port* LAN. Pada *controller* Opendaylight akan dimasukan script ACL berbasis XML (*eXtensible Markup Language*) agar kita dapat membatasi hak akses client kepada *port* TCP *(Transmission Control Protocol)* dari PC *server* 1 dan 2. PC *server* yang masing-masing diinstall WEB dan FTP server akan terhubung langsung ke *switch* Raspberry Pi, begitu pula dengan *client* akan terhubung ke *switch*. Pemberian alamat IP pada setiap perangkat akan dilakukan secara static dan berbeda satu sama lain sehingga kita dapat memastikan bahwa setiap perangkat telah mendapatkan IP-nya masing-masing. Selanjutnya adalah perancangan ACL yang akan diterapkan pada *controller*, seperti yang ada pada Tabel 1.

TABEL I
PERANCANGAN ACL

Nama	IP Address	Port TCP	Client dapat Mengkakses Port	Client tidak dapat Mengakses Port
Server 1	192.168.11.51	21, 80	21	80
Server 2	192.168.11.11	21, 80	80	21
Client	192.168.11.2	-	-	-

Dalam Tabel 1, dapat dilihat terdapat 2 *server* terdaftar dalam ACL. Pada *server* 1 dengan IP 192.168.11.51 *port* TCP yang di-*block* untuk diakses oleh komputer *client* adalah *port* 80 yang merupakan *port* HTTP *(HyperText Transfer Protocol)* dan *port* yang diizinkan adalah *port* 21 yang merupakan *port* FTP *(File Transfer Protocol)*. Berbeda dengan *server* 1, *server* 2 dengan IP 192.168.11.1 *port* yang diblok adalah *port* 21 dan *port* yang diizinkan adalah *port* 80.

Implementasi: Pada tahap ini akan dilakukan pengimplementasian konfigurasi protocol *openflow* ke dalam *switch* dan pengimplementasian fungsi ACL ke dalam *controller* Opendaylight. Dalam konfigurasi *switch*, dilakukan penginstallan OpenvSwitch pada Raspberry Pi dan diberikan IP *static* pada eth0 atau *interface* utama pada Raspberry Pi, kemudian dilanjutkan pengaturan untuk pengalokasian setiap *port* LAN pada Raspberry Pi dengan membuat protokol *openflow*.

Pertama-tama akan dibuat *bridge* dengan nama br0 kemudian secara otomatis *interface* eth0 akan digunakan untuk *port* LAN pertama dan IP *address* yang digunakan sama dengan IP *address static* yang telah dikonfigurasi sebelumnya pada Raspberry Pi. Setelah membuat *bridge*, *port* LAN tambahan akan dialokasikan kepada *bridge* br0 dan menjadi *port* Openflow sehingga nantinya perangkat yang tersambung pada *port* LAN tersebut dapat terdeteksi oleh *controller* Opendaylight. Setelah menambahkan semua *port* LAN pada bridge maka selanjutnya akan ditambahkan IP *controller* yang terhubung pada eth0. *Openflow* akan menggunakan protokol TCP sebagai *transport protocol* dengan *port* 6633 yang menjadi tempat lalu lintas dari traffic Openflow. Hasil dari konfigurasi *switch* dapat dilihat pada Tabel 2.

TABEL II
KONFIGURASI SWITCH RASPBERRY PI

Bridge	Type	Port Bridge	Interface	Controller
br0	internal	br0	eth0	-
br0	-	eth1	eth1	-
Bro	-	eth2	eth2	-
br0	-	eth3	eth3	-
br0	-	-	-	192.168.0.2:6633

Dalam Tabel 2 terlihat bahwa setiap *interface* yang ada berada dalam satu *bridge* yang sama yaitu br0. Pada bagian *port bridge* dapat dilihat *port* br0 berbeda dengan *port-port* yang lain. Hal itu dikarenakan *port* tersebut diberikan secara otomatis saat melakukan konfigurasi awal pengalokasian pada *bridge* br0. Port br0 secara *default* mendapatkan *interface* eth0 karena eth0 merupakan *interface default* pada perangkat *switch* Raspberry Pi. Kemudian untuk tiga *interface* terakhir digunakan untuk tiga perangkat yang akan tersambung ke *switch*. Terakhir pada table konfigurasi *switch* dapat dilihat pada kolom *controller*, IP 192.168.0.2:6633 yang merupakan IP dari *controller* ditambahkan pada *bridge* br0 sehingga *switch* Raspberry Pi terhubung kepada *controller* Opendaylight

Pengujian Sistem: Tahap pengujian sistem dilakukan dengan satu *client* yang diberikan IP *address* secara *static* dan sudah terhubung pada infrastruktur jaringan yang telah diberikan ACL melalui *controller*. Kemudian *client* tersebut akan mencoba mengakses masing-masing masing-masing layanan WEB *server* dan FTP *server* pada kedua *server*, sehingga dapat diketahui apakah penerapan ACL pada infrastruktur jaringan telah berhasil atau tidak. Di dalam lingkungan SDN, pemfilteran *packet* dilakukan dengan metode *flow-control*, dimana metode ini menjamin entitas pengirim tidak akan membanjir data kepada entitas penerima. Selain itu, *flow-control* dapat pula mengatur *policy* dan *flow* dari data-data yang akan berlalu-lintas pada lingkungan SDN. Karena *openflow switch* bekerja pada Layer 2 dalam OSI Layer, maka diperlukan parameter-parameter yang mampu menyokong penerapan ACL dengan menggunakan metode *flow-control*. Pada tahap Pengujian

JUTEI Edisi Volume.2 No.2 Oktober 2018
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.2018.22.100

Sistem akan dilakukan beberapa pengujian saat sebelum dan sesudah ditambakannya ACL ke dalam Server 1 dan Server 2. Dalam melakukan pengujian akses ke WEB *server*, digunakan aplikasi *web browser* Google Chrome sebagai media untuk mengetahui apakah akses kepada WEB *server* berhasil atau tidak. Dengan memasukkan alamat IP dari Server 1 dan Server 2, maka dapat diketahui WEB *server* mana yang di-*block* dan mana yang tidak. Kemudian untuk pengujian akses layanan FTP *server* digunakan aplikasi WinSCP. Dengan menentukan *port* FTP dan memasukan alamat IP dari Server 1 dan Server 2 dalam 2 kali percobaan, dapat diketahui apakah akses kepada FTP *server* berhasil atau tidak. Lalu untuk pengujian *flow* data yang ada infrastruktur jaringan SDN digunakan aplikasi OpenFlow Manager (OFM). OFM merupakan aplikasi tambahan untuk mempermudah kinerja *controller* di dalam infrastruktur jaringan SDN.

IV. HASIL DAN ANALISIS

Penerapan konsep SDN didasarkan pada bagaimana melakukan manajemen sebuah infrastruktur jaringan dengan menggunakkan aplikasi *controller*sebagai pusat kontrol bagi *client-client* yang berada pada infrastruktur jaringan tersebut. Hasil dari simulasi konsep SDN ini adalah penerapan ACL ke dalam infrastruktur jaringan seperti yang terlihat pada Gambar 5 di atas. ACL secara umum adalah sebuah metode untuk memfilter *packet-packet* yang keluar dan masuk dalam sebuah jaringan.

Dalam infrastruktur jaringan SDN yang dibuat berdasarkan Gambar 5, dapat dilihat unsur penentu dari adanya sebuah infrastrukur jaringan SDN adalah *controller*, *openflow switch*, dan tentunya perangkat-perangkat jaringan yang akan terhubung satu dengan yang lainnya. Tentunya setelah semua unsur tersebut terhubung, tidak mungkin jika *controller* dapat langsung mendeteksi *openflow switch* dan perangkat jaringan lainnya, Awalnya diperlukan pemberian protokol pada jalur yang menghubungkan antara *openflow switch* dengan *controller* dan pada akhirnya perlu ada alokasi pada setiap *port* LAN yang akan menghubungkan perangkat jaringan dengan *openflow switch*. Ada beberapa *flow* default yang digunakan untuk menghubungkan setiap perangkat jaringan ke *openflow switch* dan dari *openflow switch* menuju ke *controller*. Berikut adalah Kode Program 1, dimana akan ditampilkan list dari *flow default* sebelum dilakukannya pengujian penerapan ACL melalui *policy* dan *flow* tambahan. *Flow default* tersebut menjadi alasan bagaimana setiap unsur jaringan tersebut dapat terhubung.

```
1. cookie=0x2b00000000000000, duration=6812.268s, table=0,
   n_packets=8, n_bytes=480, priority=100,dl_type=0x88cc
   actions=CONTROLLER:65535
2. cookie=0x2b00000000000000, duration=6811.179s, table=0,
   n_packets=5593, n_bytes=2564205, priority=2,in_port=3
   actions=output:2,output:1,CONTROLLER:65535
3. cookie=0x2b00000000000001, duration=6811.175s, table=0,
   n_packets=108426, n_bytes=140330778, priority=2,in_port=2
   actions=output:3,output:1,CONTROLLER:65535
4. cookie=0x2b00000000000002, duration=6811.155s, table=0,
   n_packets=66004, n_bytes=9459241, priority=2,in_port=1
   actions=output:3,output:2,CONTROLLER:65535
5. cookie=0x2b00000000000000, duration=6812.269s, table=0,
   n_packets=0, n_bytes=0, priority=0 actions=drop
```

Kode Program. 1. List Flow Default

Pada Kode Program 1, terdapat 5 *list flow* yang terdapat di dalam *flow-table* dari perangkat Raspberry Pi yang telah dijadikan *openflow switch*. Di dalam *flow table* tersebut dapat terlihat banyak parameter-parameter sebagai penunjang dari adanya sebuah *flow*. Mulai dari cookie, duration, table, n_packet, n_bytes, priority, dl_type, in_port, dan sebagainya. Cookie merupakan identitas dari sebuah *flow* yang ada di dalam *flow table* itu sendiri. Duration merupakan jumlah detik atau jumlah waktu dari *flow* yang telah aktif. Kemudian untuk n_bytes dan n_packet merupakan jumlah *packet* dan jumlah *bytes* yang cocok pada *flow* yang masuk ataupun keluar. Sedangkan priority adalah prioritas yang diberikan *openflow switch* kepada *flow-flow* yang masuk. Untuk dl_type itu sendiri digunakan untuk mengindikasikan protokol apa yang akan dienkapsulasi di dalam *flow*. Pada bagian action, digunakan untuk menentukan kemana arah *flow* ataupun untuk mengatur *flow* apa saja dapat masuk di dalam infrastruktur jaringan SDN.

Setiap *flow* yang berada di dalam *list flow* atau *flow table* memiliki fungsinya masing-masing. *Flow* baris 1 pada Kode Program 1 menjelaskan bahwa semua *flow* yang melewati OpenFlow *switch* akan diatur oleh *contoller* yang berkomunikasi di protokol *port* nomor 65535. *Flow* baris 2 pada Kode Program 1 menjelaskan bahwa semua *flow* yang masuk dari *port* fisik *switch* 3 akan diperbolehkan untuk diteruskan ke *port* fisik *switch* 1 dan *port* fisik *switch* 2. *Flow* baris 3 pada Kode Program 1 menjelaskan bahwa semua *flow* yang masuk dari *port* fisik *switch* 2 akan diperbolehkan untuk diteruskan ke *port* fisik *switch* 3 dan *port* fisik *switch* 1. *Flow* baris 4 pada Kode Program 1 menjelaskan bahwa semua *flow* yang masuk dari *port* fisik *switch* 1 akan diperbolehkan untuk diteruskan ke *port* fisik *switch* 2 dan *port* fisik *switch* 3. *Flow* baris 5 pada Kode Program 1 menjelaskan bahwa semua *flow* yang memiliki prioritas 0 tidak diperbolehkan lewat atau "*drop*", *flow* ini biasanya muncul sebagai akibat dari paket sampah.

```
{
   "flow": [
      {
         "table_id": 0,
         "id": "Block WEB Server",
         "priority": 900,
         "hard-timeout": 0,
         "idle-timeout": 0,
         "match": {
            "ethernet-match": {
               "ethernet-type": {
                  "type": 2048
               }
            },
            "ipv4-destination": "192.168.11.51/32",
            "ip-match": {
               "ip-protocol": 6
            },
            "tcp-destination-port": 80
         },
         "instructions": {
            "instruction": [
               {
                  "order": 0,
                  "apply-actions": {
                     "action": [
                        {
                           "order": 0,
                           "drop-action": {}
                        }
                     ]
                  }
               }
            ]
         }
      }
   ]
}
```

Kode Program. 2. Block WEB Server

Dalam pemberian *policy* dan *flow* pada infrastruktur jaringan SDN menggunakan *controller* Opendaylight akan melakukan *push configuration* atau mengirim konfigurasi mengenai *policy* dan *flow* yang telah ditetapkan ke setiap perangkat yang terhubung melalui protokol *openflow*. Dalam penelitian ini, *policy* dan *flow* yang telah ditetapkan tersebut diterapkan pada sebuah kode program berbasis XML yang dimasukkan pada *controller* Opendaylight. Berikut adalah Kode Program 1 yang digunakan untuk membatasi akses *client* kepada Server 1.

Pada Kode Program 2, dijelaskan bahwa pertama-tama akan dibentuk sebuah *flow* dengan pendeklarasian awal untuk table_id, id, priority, hard-time-out, dan idle-timeout. Pada table_id diberikan nilai 0 karena table *flow* yang ada secara default adalah 0. Sedangkan untuk id dibuat untuk mengidentifikasi nama *flow* dan *policy* yang berjalan di dalam *controller*. Lalu, untuk priority diberikan nilai 1000 karena untuk membedakan dari *flow-flow* lainnya, semakin tinggi nilai prioriy, maka *flow* tersebut yang lebih dahulu dicocokan dengan aliran data yang masuk. Jika terdapat *flow* dengan priority yang sama, maka salah satu *flow* tidak dapat dieksekusi. Kemudian untuk hard-timeout dan idle-timeout diberikan nilai *default* 0 yang berarti tidak ada *timeout* yang diberikan untuk *flow* yang cocok dengan parameter. Pemberian *policy* pada *flow* dilakukan pada bagian ini, dimana bagian match akan diisi tipe *ethernet* yang digunakan dan untuk protokol jaringan TCP/IP diisikan tipe 2048 sehingga protokol IPv4 dapat berjalan. Setelah pemberian tipe *ethernet*, diisikan alamat pada ipv4-destination dengan alamat dari Server 1 yaitu 192.168.11.51/32 dan ip-protokol untuk IPv4 adalah 6. Dalam pendeklarasian IP *address client* pada ipv4-destination perlu menggunakan *subnet* /32, karena *subnet* tersebut digunakan untuk mengindikasikan bahwa IP *address client* tersebut berdiri sendiri atau disebut dengan *single link*. Jika ipv4-destination diisi dengan *subnet* /24 atau dengan *subnet* lainnya maka yang terjadi adalah *openflow switch* akan membaca IP *network* dari IP *address* dan *subnet* tersebut. Setelah parameter-parameter tersebut diisi, maka hal terakhir yang perlu dilakukan adalah menambahkan *action* drop, jadi *action* drop berguna untuk memfilter *flow-flow* yang mengarah pada *port* 80 yang juga bisa disebut dengan HTTP/WEB *server* yang ada pada Server 1.

Setelah dilakukannya penambahan *flow* untuk membatasi akses *client* kepada WEB Server, maka dapat terlihat adanya penambahan *flow* yang ada pada *list flow* di dalam *openflow switch* seperti yang tertera pada Kode Program 3 berikut.

```
1. cookie=0x0, duration=2534.004s, table=0, n_packets=79,
   n_bytes=8384,
   priority=900,tcp,nw_dst=192.168.11.51,tp_dst=80
   actions=drop
2. cookie=0x2b00000000000000, duration=6912.268s, table=0,
   n_packets=8, n_bytes=480, priority=100,dl_type=0x88cc
   actions=CONTROLLER:65535
3. cookie=0x2b00000000000000, duration=6911.179s, table=0,
   n_packets=5593, n_bytes=2564205, priority=2,in_port=3
   actions=output:2,output:1,CONTROLLER:65535
4. cookie=0x2b00000000000001, duration=6911.175s, table=0,
   n_packets=108426, n_bytes=140330778, priority=2,in_port=2
   actions=output:3,output:1,CONTROLLER:65535
5. cookie=0x2b00000000000002, duration=6911.155s, table=0,
   n_packets=66004, n_bytes=9459241, priority=2,in_port=1
   actions=output:3,output:2,CONTROLLER:65535
6. cookie=0x2b00000000000000, duration=6912.269s, table=0,
   n_packets=0, n_bytes=0, priority=0 actions=drop
```

Kode Program. 3. List Flow setelah penambahan flow Block WEB Server

List flow pada Kode Program 3 sedikit berbeda dari Kode Program 1, karena adanya penambahan *flow* Block WEB Server yang telah ditambahkan dengan menggunakan Kode Program 2. Dapat terlihat pada *flow* 1 pada *list flow* memiliki parameter-parameter yang berbeda seperti tcp, nw_dst, tp_dst. Deklarasi dari variable tcp merupakan deklarasi jalur IP yang digunakan oleh IP *address client*, seperti Kode Program 2 terdapat pendeklarasian ip-protokol dengan nilai 6, nilai 6 tersebut adalah nilai untuk jalur TCP. Untuk deklarasi dari nw_dst digunakan sebagai untuk mengetahui *network* atau IP *address* tujuan untuk menandakan bahwa efek dari *flow* yang dibuat akan mengarah pada IP *address* tersebut. Sedangkan untuk tp_port atau yang dimaksud dengan TCP *port destination* digunakan untuk menentukan parameter dari *port* TCP yang akan

JUTEI Edisi Volume.2 No.2 Oktober 2018
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.2018.22.100

digunakan oleh *flow* untuk membatasi akses *client* ke *port* HTTP dari Server 1. Setelah setiap parameter diberikan nilai sesuai perencanaan sebelumnya maka akan diterapkan sebuah *action* drop, karena *action* drop berguna untuk menghentikan *packet-packet* yang mengalir pada *flow* di dalam *openflow switch*. *Packet* yang dihentikan pun menyesuaikan dengan parameter yang ditentukan sebelumya, jika ingin menghentikan akses *packet* ke arah WEB Server maka *port* HTTP dari *server* tersebut dimasukkan ke dalam parameter sehingga *packet* yang dihentikan hanyalah mengarah pada *port* HTTP saja, tidak mempengaruhi akses *port-port* yang lain seperti FTP, SSH, Telnet, dan sebagainya. Tidak ada perbedaan dari Kode Program 1 untuk *flow-flow* lainnya. Hanya ada penambahan duration pada 5 *flow* sebelumnya, karena kelima *flow* tersebut lebih lama aktif daripada *flow* yang baru ditambahkan.

Selanjutnya adalah bentuk dari kode program yang digunakan untuk membatasi akses *client* kepada Server 2. Kode program tersebut akan ditampikan Kode Program 4. Seperti yang dapat dilihat pada Kode Program 2, Kode Program 4 tidak jauh berbeda untuk struktur pendeklarasiannya. Beberapa hal yang berbeda dari Kode Program 2 adalah deklarasi priority diberikan nilai 900, *policy* dari ipv4-destination atau alamat IP tujuan diisi dengan alamat dari Server 2 yaitu 192.168.11.11/32 dan ip-protokol tetap diisi dengan nilai 6. Deklarasi nilai *priority* dibedakan karena *openflow switch* tidak dapat mengeksekusi *flow* yang memiliki nilai *priority* yang sama. Sedangkan untuk *port* TCP diisi dengan nilai untuk *port* FTP yaitu 21. Kemudian yang terakhir dideklarasikan *action* drop untuk membatasi *client* dalam mengakses FTP *server* dari Server 2. Seteleh melakukan penambahan *flow* melalui *controller* dengan menggunakan Kode Program 4, maka akan bertambah juga *flow* baru kedalam *list flow* yang di dalam *openflow switch* yang ditampilkan pada Kode Program 5.

Kode Program 5 menampilkan *list flow* baru dengan adanya penambahan *flow* Block FTP Server. Beberapa hal yang berbeda dari *list flow* tersebut dengan *list flow* pada Kode Program 3 adalah adanya *flow* baru yang digunakan untuk membatasi akses *client* dalam mengakses *port* FTP dari Server 2, kemudian perbedaan dari *flow* 1 dan *flow* 2 adalah deklarasi IP *address* tujuan dan *port* TCP tujuan, dan yang terakhir adalah duration dari setiap *flow* bertambah karena *flow* yang lain lebih lama aktif. *Flow* baris 1 pada Kode Program 5 menjelaskan bahwa semua *flow* yang menuju ke alamat IP 192.168.11.11 dan berkomunikasi pada nomor protokol tcp *port* 21 akan di *drop*, nilai prioritas 1000 menyebabkan *flow* 1 akan dieksekusi terlebih dahulu dari *flow* yang lain. *Flow* baris 2 pada Kode Program 5 menjelaskan bahwa semua *flow* yang menuju ke alamat IP 192.168.11.51 dan berkomunikasi pada nomor protokol tcp *port* 80 akan di *drop*, nilai prioritas 900 menyebabkan *flow* berada diurutan 2 karena ada *flow* 1 yang memiliki prioritas lebih besar. Nilai prioritas yang semakin besar akan membuat flow berada di posisi tinggi dan akan dieksekusi terlebih dahulu.

```
1.  {
2.    "flow": [
3.      {
4.        "table_id": 0,
5.        "id": "Block FTP Server",
6.        "priority": 1000,
7.        "hard-timeout": 0,
8.        "idle-timeout": 0,
9.        "match": {
10.         "ethernet-match": {
11.           "ethernet-type": {
12.             "type": 2048
13.           }
14.         },
15.         "ipv4-destination": "192.168.11.11/32",
16.         "ip-match": {
17.           "ip-protocol": 6
18.         },
19.         "tcp-destination-port": 21
20.       },
21.       "instructions": {
22.         "instruction": [
23.           {
24.             "order": 0,
25.             "apply-actions": {
26.               "action": [
27.                 {
28.                   "order": 0,
29.                   "drop-action": {}
30.                 }
31.               ]
32.             }
33.           }
34.         ]
35.       }
36.     }
37.   ]
38. }
```

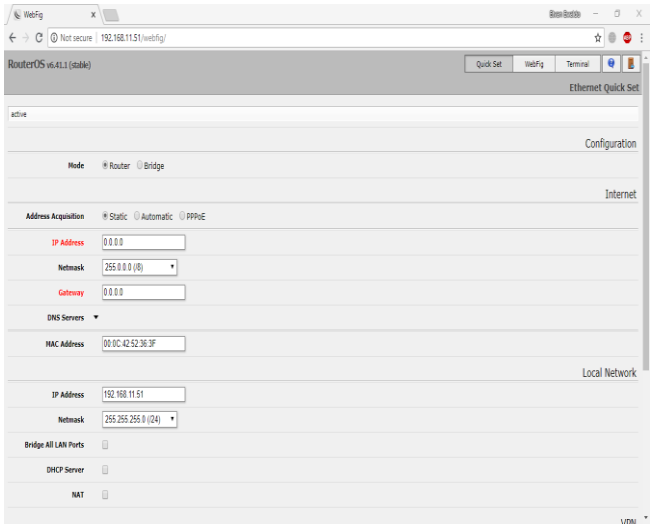
Kode Program. 4. Block FTP Server

Hasi dari penerapan ACL melalui *controller* pada Kode Program 2 dan Kode Program 4 adalah *client* yang ada tidak dapat mengakses layanan HTTP/WEB Server yang berada pada Server 1 dan tidak dapat mengakses layanan dari FTP *server* pada Server 2. Berikut adalah gambar dimana sebelum diberikannya Kode Program 1 pada *controller* yang ditampilkan pada Gambar 6.

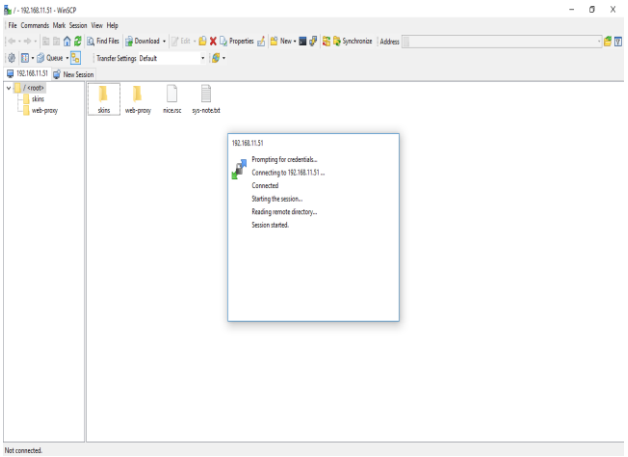
Kode Program. 5. List Flow setelah penambahan flow Block FTP Server

```
1. cookie=0x0, duration=3978.785s, table=0, n_packets=12,
   n_bytes=792,
   priority=1000,tcp,nw_dst=192.168.11.11,tp_dst=21
   actions=drop
2. cookie=0x0, duration=4534.004s, table=0, n_packets=79,
   n_bytes=8384,
   priority=900,tcp,nw_dst=192.168.11.51,tp_dst=80
   actions=drop
3. cookie=0x2b00000000000000, duration=7012.268s, table=0,
   n_packets=8, n_bytes=480, priority=100,dl_type=0x88cc
   actions=CONTROLLER:65535
4. cookie=0x2b00000000000000, duration=7011.179s, table=0,
   n_packets=5593, n_bytes=2564205, priority=2,in_port=3
   actions=output:2,output:1,CONTROLLER:65535
5. cookie=0x2b00000000000001, duration=7011.175s, table=0,
   n_packets=108426, n_bytes=140330778, priority=2,in_port=2
   actions=output:3,output:1,CONTROLLER:65535
6. cookie=0x2b00000000000002, duration=7011.155s, table=0,
   n_packets=66004, n_bytes=9459241, priority=2,in_port=1
   actions=output:3,output:2,CONTROLLER:65535
7. cookie=0x2b00000000000000, duration=6812.269s, table=0,
   n_packets=0, n_bytes=0, priority=0 actions=drop
```

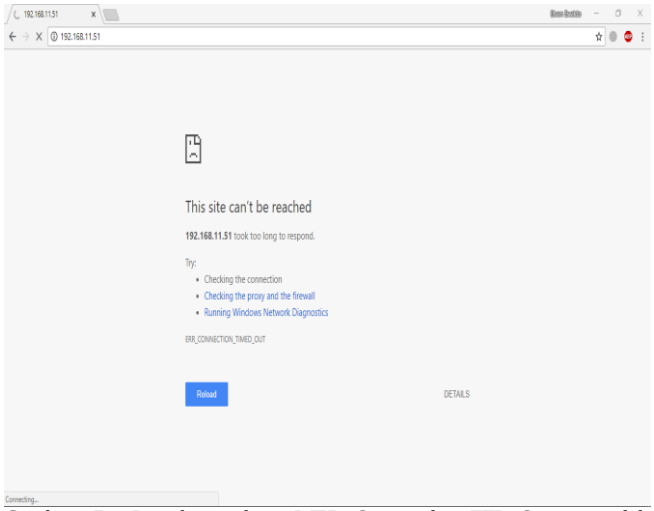
Dapat terlihat bahwa Gambar 6 menampilkan hasil percobaan pertama *client* yang mencoba mengakses HTTP Server dan FTP Server pada Server 1 dan hasilnya berhasil. Akses HTTP server dikatakan berhasil dikarenakan pada Gambar 6a muncul halaman *web default* dari HTTP server yang berupa tampilan *web default* Mikrotik, sedangkan akses FTP server dikatakan berhasil ketika pada saat aplikasi WinSCP dijalankan dan menuju ke alamat FTP server, pada kolom status koneksi akan muncul informasi *"authenticated"* seperti yang ditunjukkan pada Gambar 6b. Lalu kemudian, setelah ACL diterapkan melalui Kode Program 1 yang ditambahkan ke dalam *controller* Opendaylight, maka hasilnya akan seperti pada Gambar 7. Setelah dilakukan penerapan ACL pada *controller* Opendaylight, maka dapat terlihat hasilnya seperti Gambar 7. *Client* tidak dapat mengakases WEB Server karena *port* HTTP dari Server 1 telah tutup. Sedangkan untuk *port* FTP masih dapat diakses, dikarenakan tidak ada *policy* atau ACL yang diterapkan untuk menghentikan akses *port* FTP pada Server 1.



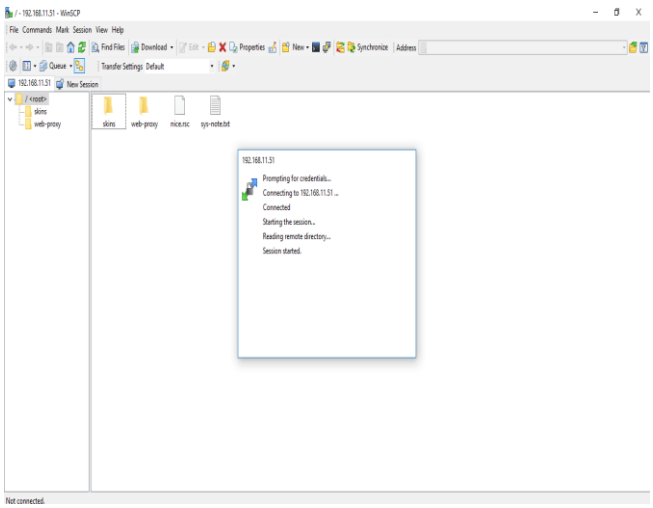
Gambar. 6a. Percobaan Akses HTTP Server dan FTP Server sebelum penambahan kode program Block WEB Server pada Server 1



Gambar. 6b. Percobaan Akses HTTP Server dan FTP Server sebelum penambahan kode program Block WEB Server pada Server 1



Gambar. 7a. Percobaan akses WEB Server dan FTP Server setelah penambahan kode program Block WEB Server pada Server 1



Gambar. 7b. Percobaan akses HTTP Server dan FTP Server setelah penambahan kode program Block HTTP Server pada Server 1

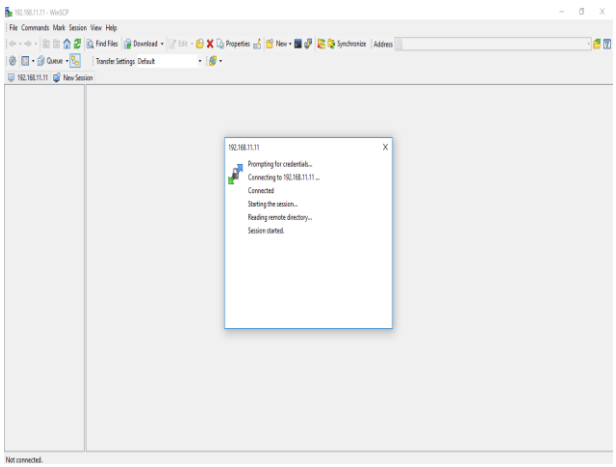
JUTEI Edisi Volume.2 No.2 Oktober 2018
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.2018.22.100

Pada percobaan ini akses ke HTTP Server dikatakan gagal seperti pada Gambar 7a dibuktikan dengan munculnya pesan *"error"* pada halaman web, yang sebelumnya pada Gambar 6a dapat muncul halaman *web default* Mikrotik. Pada Kode Program 6 nilai nw_dst=192.168.11.51 dan tp_dst=80 dengan actions=drop yang menyebabkan *"error"* ini terjadi, sebab alamat IP Server HTTP dan nomor protokol port HTTP cocok dengan definisi *flow-control* ini. Akses ke FTP server pada gambar 7b juga dikatakan berhasil dengan adanya informasi *"authenticated"* pada kolom status koneksi aplikasi WinSCP.

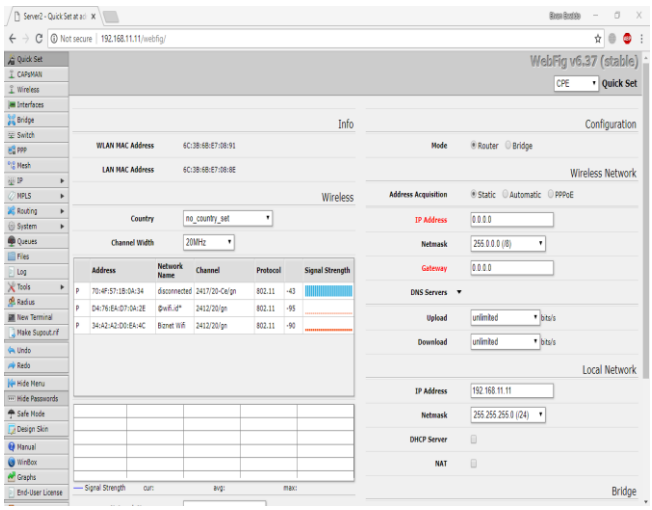
```
1.  cookie=0x0, duration=4534.004s, table=0, n_packets=79,
    n_bytes=8384,
    priority=900,tcp,nw_dst=192.168.11.51,tp_dst=80
    actions=drop
```

Kode Program. 6. *Flow-control* untuk block akses WEB Server

Selanjutnya pada percobaan selanjutnya, akan dilakukan penerapan ACL pada Server 2. Sebelum itu, akan diuji bagaimana respon dari HTTP Server dan FTP Server pada Server 2 sebelum diberikan ACL melalui Kode Program 2 kepada *controller* Opendaylight. Berikut adalah hasil uji akses Server 2, dapat dilihat pada Gambar 8.



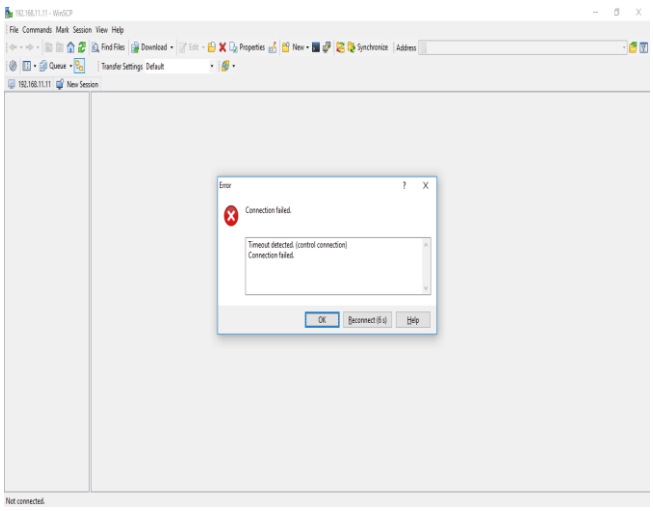
Gambar. 8a. Percobaan akses FTP Server dan HTTP Server sebelum penambahan kode program Block FTP Server pada Server 2



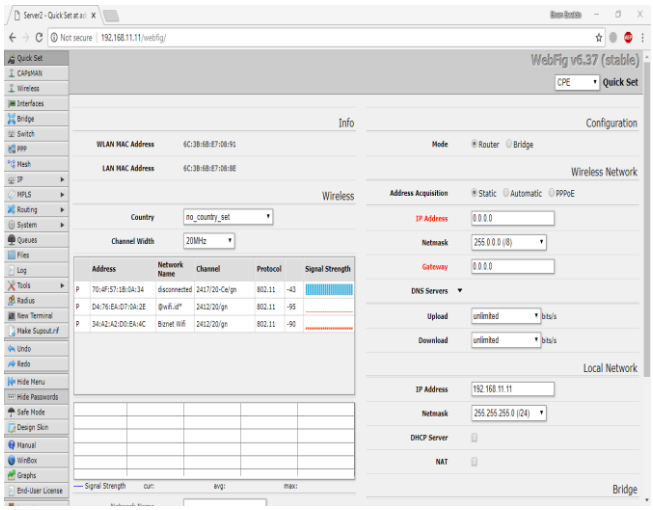
Gambar. 8b. Percobaan akses FTP Server dan HTTP Server sebelum penambahan kode program Block FTP Server pada Server 2

Dapat terlihat pada Gambar 8, bahwa akses terhadap HTTP Server dan FTP Server, telah berhasil dilakukan. Kemudian selanjutnya, dilakukan penerapan ACL untuk mem-*block* akses *port* FTP melalui Kode Program 2. Hal yang serupa dengan pengujian pada Server 1, Server 2 pada akses HTTP Server pada Gambar 8a dan akses FTP Server pada Gambar 8b semua berhasil dengan munculnya halaman *web default* Mikrotik dan informasi *"authenticated"* pada aplikasi WinSCP.

Berikut adalah hasil percobaan akses ke WEB Server dan FTP Server dari Server 2, dapat dilihat pada Gambar 9. Gambar 9a menunjukkan akses FTP Server pada kolom status koneksi aplikasi WinSCP menampilkan pesan *"error"* akibat dari penerapan *flow-control* yang dilakukan. Sedangkan akses HTTP Server pada Gambar 9b masih berhasil dengan munculnya tampilan *web default* Mikrotik.



Gambar. 9a. Percobaan akses FTP Server dan HTTP Server sesudah penambahan kode program Block FTP Server pada Server 2



Gambar. 9b. Percobaan akses FTP Server dan HTTP Server sesudah penambahan kode program Block FTP Server pada Server 2

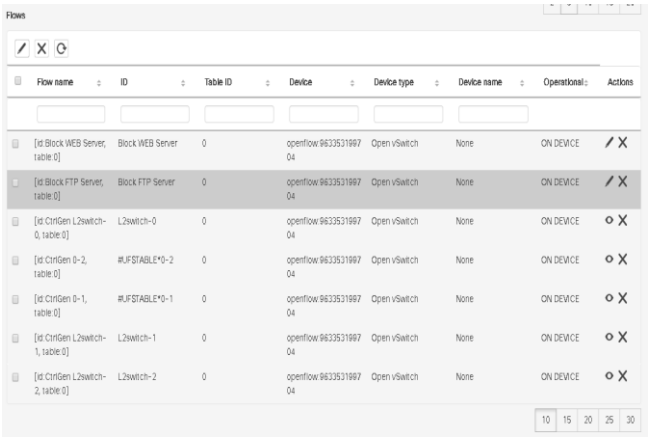
Setelah diterapkan ACL untuk melakukan *block* pada *port* FTP, dapat terlihat pada Gambar 9 bahwa FTP Server dari Server 2 gagal diakses, sedangkan HTTP Server masih dapat diakses. Pada Kode Program 7 nilai nw_dst=192.168.11.11 dan tp_dst=21 dengan actions=drop

yang menyebabkan *"error"* akses FTP Server terjadi, sebab alamat IP Server FTP dan nomor protokol port FTP cocok dengan definisi *flow-control* ini

```
1.  cookie=0x0, duration=3978.785s, table=0, n_packets=12,
    n_bytes=792,
    priority=1000,tcp,nw_dst=192.168.11.11,tp_dst=21
    actions=drop
```

Kode Program. 7. *Flow-control* untuk block akses FTP Server

Dapat dilihat dari kedua percobaan diatas, bahwa penerapan ACL pada konsep SDN menggunakan Raspberry Pi berhasil dilakukan, dengan menggunakan *controller* Opendaylight yang melakukan *push-configuration* kepada *switch openflow* Raspberry Pi. Pemberian nilai prioritas pada penelitian ini menunjukkan bahwa *flow-control* untuk blok akses HTTP Server memiliki nilai yang lebih besar daripada nilai prioritas dari *flow-control* untuk blok akses FTP Server. Nilai prioritas 1000 pada *flow-control* HTTP dan 900 pada *flow-control* FTP menyebabkan jika ada 2 *flow* yang pada satu kemungkinan dating bersamaan, maka *controller* akan memproses terlebih dahulu *flow-control* HTTP yang memiliki nilai 1000. Kegunaan lain dari pemberian nilai prioritas ini adalah jika akan dikaitkan dengan manajemen antrian *flow* yang dapat berdampak pada kualitas kecepatan respon jaringan secara luas.



Gambar. 10. Tampilan *flow list* dari aplikasi OFM

Gambar 10 menunjukkan hasil dari manipulasi *flow* yang sudah dilakukan, tampilan OFM dapat mempermudah visualisasi terhadap *flow-control* yang dilakukan, karena OFM berbasis *web*. Informasi yang terdapat di OFM akan sama dengan *flow-list* yang ada pada *controller* Opendaylight, karena manipulasi *flow* akan diteruskan ke perangkat Openflow *switch* untuk dieksekusi di level perangkat. Konfigurasi ACL pada *flow-control* dapat dilakukan langsung dari *controller* Opendaylight atau dapat menggunakan aplikasi lain yang berbasis RESTCONF API. RESTCONF API biasa digunakan untuk komunikasi antara kode program aplikasi ke *controller* Opendaylight yang berisi parameter-parameter *flow* dan nantinya akan diteruskan keperangkat switch oleh protokol Openflow.

V. KESIMPULAN

Berdasarkan kedua penerapan ACL yang dilakukan dapat ditarik kesimpulan bahwa ketika perangkat-perangkat jaringan di dalam infrastruktur jaringan SDN terhubung ke *controller*, maka protokol *openflow* dapat digunakan sebagai jalur komunikasi antara perangkat jaringan dengan *controller* atau yang disebut sebagai *southbound interface*. Ketika akan melakukan penerapan ACL melalui *flow-control*, *controller* perlu diberikan parameter-parameter yang jelas sehingga *controller* dapat melakukan *push-configuration* kepada *switch openflow* Raspberry Pi. Penerapan parameter-parameter ACL dapat dilakukan melalui *controller* langsung atau menggunakan aplikasi lain yang berbasis RESTCONF API atau yang disebut sebagai *northbound interface*. Setelah ACL berhasil diterapkan, maka *client* dapat langsung terkena dampk dari penerapan ACL tersebut, yaitu *client* tidak dapat mengakses *port* HTTP dari Server 1 dan *client* tidak dapat mengakses *port* FTP dari Server 2. Dalam hal ini *controller* dapat melakukan penerapan-penerapan lain yang berhubungan dengan manajemen jaringan dalam lingkungan SDN, bergantung pada layanan apa yang dibutuhkan.

Pada penelitian ini tidak terlepas dari kekurangan yang kemungkinan dapat di sempurnakan pada penelitian lain, ada beberapa saran yang bisa dijadikan sebagai tambahan seperti dapat dilakukannya pengujian langsung untuk lingkup SDN yang lebih luas, menambah jumlah *device* yang digunakan, mencoba menggunakan *switch* sesungguhnya, dan mencoba melakukan penerapan-penerapan lain dalam lingkungan SDN.

DAFTAR PUSTAKA

[1] M. Erel, E. Toeman, Y. Özçevik, G. Seçinti, B. Canberk. (November 2015). "Scalability analysis and flow admission control in mininet-based SDN environment". 2015 *IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*. [Online]. Tersedia: https://ieeexplore.ieee.org/document/7387396/

[2] Simamora, S.N.M.P., Hendrarini, N., dan Lya Umi Sitepu, E. (Mei 2011). Metode Access Control List sebagai Solusi Alternatif Seleksi Permintaan Layanan Data pada Koneksi Internet. *Jurnal Teknologi Informasi Politeknik Telkom* [Online]. *1(1)*, hal. 15. Tersedia: http://journals.telkomuniversity.ac.id/jti/article/view/414

[3] Ariman, M., Seçinti, G., Erel, M., and Canberk, B. (2015). Software Defined Wireless Network Testbed using Raspberry Pi OF Switches with Routing Add-on. Dipresentasikan di 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN). [Online]. Tersedia: https://ieeexplore.ieee.org/document/7387397/

[4] "Developer's guide - basic approach to openwrt," OpenWrt - Wireless Freedom, Tech. Rep., (2012). [Online]. Tersedia: http://wiki.openwrt.org/doc/techref/externaldocumentation

[5] Ahurra, A., Kamurali, K.M., O'Briain, D., and Okello, D. (April 2017). Software Defined Networking (SDN). Dipresentasikan di ICICT 2017 ISBAT University. [Online]. Tersedia: https://www.researchgate.net/publication/316666791_Software_Defined_Networking_a_comparison_with_the_Legacy_network

[6] Alsmadi, I. (Februari 2016). The integration of access control levels based on SDN. *Int. J. High Performance Computing and Networking*. [Online]. *9(4)*, hal. 286-287. Tersedia: https://www.researchgate.net/publication/290438041_The_integration_of_access_control_levels_based_on_SDN

[7] Mulyana, E.(2015) Buku Komunitas SDN-RG. [Online]. Tersedia: https://eueung.gitbooks.io/buku-komunitas-sdn-rg/content/

[8] Basit, A., Qaisar, S., Hamid Rasool, S., dan Ali, M. (Maret 2017). SDN Orchestration for Next Generation Inter-Networking: A Multipath Forwarding Approach. *IEEE Access*. [Online]. Vol. 5, hal. 13077–13089. Tersedia: https://ieeexplore.ieee.org/document/7879870/

JUTEI Edisi Volume.2 No.2 Oktober 2018
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.2018.22.100

[9] (2012) SDxCentral website. [Online].Tersedia: https://www.sdxcentral.com/
[10] R. Toghraee. (Mei 2017). Learning OpenDaylight. (edisi pertama) [Online]. Tersedia: https://books.google.co.id/
[11] Opendaylight. (2017) Carbon is the sixth release of OpenDaylight (ODL). [Online]. Tersedia: https://www.opendaylight.org/what-we-do/current-release/carbon
[12] EduPERT. (2012) OpenFlow. [Online]. Tersedia: https://kb.pert.geant.net/PERTKB/OpenFlow