 ISSN: 1411-3201

PROTOTIPE INFRASTRUKTUR SOFTWARE-DEFINED NETWORK DENGAN PROTOKOL OPENFLOW MENGGUNAKAN UBUNTU SEBAGAI KONTROLER

Rikie Kartadie [1] , Ema Utami [2] , Eko Pramono [3]

[1,2,3] Magister Teknik Informatika STMIK AMIKOM Yogyakarta
email : rikie.kartadie@gmail.com [1], ema.u@amikom.ac.id [2], eko.p@amikom.ac.id [3]

Abstraksi

Fungsi SDN dan OpenFlow telah dipelajari dalam beberapa tahun terakhir untuk packet networks (paket jaringan), tetapi implementasinya masih terbilang jarang. Penelitian ini membangun infrastruktur Software-Defined Network dengan biaya murah, dan membandingkan hasil dari prototipe dengan hasil yang diperoleh dari simulator mininet. Penelitian melakukan uji pengiriman paket ICMP, uji pengiriman paket ICMP - VLAN, uji pengiriman paket ICMP - Routing, uji telnet dan uji respon switch terhadap kontroler. Prototipe yang dibangun telah berhasil dan mendekati hasil yang diperoleh simulator mininet.

***Kata Kunci* :**
Software-Defined Network, OpenFlow, prototipe, mininet.

Pendahuluan

Teknologi jaringan sekarang ini memiliki keterbatasan diantaranya: Kompleksitas yang mengarahkan kearah statis, Kebijakan yang tidak konsisten (berubah – ubah), Ketidakmampuan untuk diukur, dan Ketergantungan terhadap vendor. Karena keterbatasan ini, maka industri jaringan mencapai satu titik tujuan untuk meresponnya dengan diciptakan sebuah arsitektur baru yang dikenal degan arsitektur *Sofware-Defined Network*.[1]

OpenFlow adalah protokol yang relatif baru yang dirancang dan diimplementasikan di Stanford University pada tahun 2008. Protokol baru ini bertujuan untuk mengontrol *data plane* switch, yang telah dipisahkan secara fisik dari *control plane* menggunakan perangkat lunak pengendali (*Controller*) pada sebuah *server*. *Control Plane* berkomunikasi dengan *data plane* melalui protokol *OpenFlow*. Bentuk *Software-Defined Networking* (SDN) memungkinkan para peneliti, administrator dan operator untuk mengontrol jaringan mereka dengan perangkat lunak khusus dan menyediakan *Application Programming interface* (API) terhadap tabel *forwarding* dari switch dari vendor yang berbeda.[2]

Dalam perjalanannya, ada beberapa peneliti membuat prototipe dengan menggunakan NetFPGA card yang hanya memiliki 4 port dalam 1 card PCI-nya, seperti yang dilakukan oleh Naous ,J., dkk dan dengan langkah yang sama dilakukan kembali oleh Chung Yik,EE., penilitian ini memerlukan biaya tinggi dengan sclability yang rendah. Hal ini akan memberikan beban biaya yang besar, apabila ingin bereksperimen dengan SDN/*OpenFlow* dan bahkan apabila akan menerapkan SDN/*OpenFlow* pada jaringan, tentunya akan menambah beban, terlebih jika harus mengganti semua perangkat yang telah ada.[3][4]

Untuk melakukan implementasi SDN/*OpenFlow*, beberapa vendor menawarkan perangkat *dedicated OpenFlow switch* dengan harga yang tinggi, sebagai ilustrasi dari website www-304.ibm.com, diakses tanggal 23 November 2013, untuk switch IBM seri G8264 dengan jumlah port 48 port, diberi harga $29999 atau sekitar Rp. 350.000.000,- . Jauh lebih murah jika dibandingkan dengan switch tradisional (support OpenWrt) dengan kisaran harga mulai Rp.500.000,- hingga Rp. 2.000.000,- (sumber : www.bhineka.com, akses tanggal 23 November 2013), walau memang dengan harga yang tinggi tersebut tentu saja vendor menawarkan banyak fitur yang memang dikhususkan untuk menjalankan *OpenFlow*.

Untuk menjawab pertanyaan apakah mungkin SDN/*OpenFlow* diimplementasikan pada jaringan dengan perangkat yang telah ada yang belum mendukung *OpenFlow* tanpa mengganti perangkat menjadi perangkat khusus *OpenFlow*, akan dibuatkan prototipe/model *Software-Defined Network* dengan protokol *OpenFlow* dengan menggunakan switch OpenWrt sebagai *OpenFlow* switch.

Prototipe adalah model yg mula-mula (model asli) yg menjadi contoh, sedangkan model adalah sebuah representasi dari system atau proses yang ada pada dunia nyata. Sehingga bila dibuatkan model contoh kecil SDN/*OpenFlow* yang dapat mempresentasikan kondisi jaringan dengan arsitektur SDN/*OpenFlow*, maka model kecil tersebut dapat disebut sebuah prototipe.[5].

 ISSN: 1411-3201

Tinjauan Pustaka

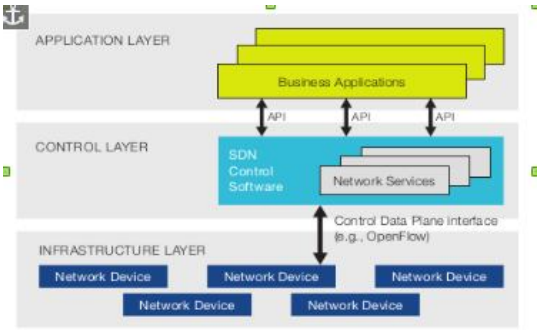
OpenFlow dapat diimplementasikan pada NetFTPGA, dijadikan sebuah *firewall* dan dapat dimonitoring menggunakan wireshark. Perbedaan mendasar pada penelitian ini adalah pada perangkat penelitian yang digunakan, penelitian ini menggunakan switch OpenWrt yang langsung terhubung dengan kontroler, sehingga bila ternyata berhasil, implementasi pada infrastruktur jaringan tidak menjadi beban biaya yang besar, karena tidak terjadi perubahan yang mendasar pada hardware infrastruktur yang telah ada. Selain itu, Chunk Yin memonitoring hasil penelitiannya menggunakan wireshark dan protokol yang di monitor adalah TCP, sedang pada penelitian ini menggunakan wireshark namun yang dimonitor adalah protokol *OpenFlow*.[4]

Jurnal Lantz Bob, dkk, melakukan percobaan tentang *prototyping* SDN dengan *OpenFlow*, namun pada jurnal ini hanya membahas tentang mininet dan pengunaannya. Mininet adalah sebuah simulator system untuk prototipe jaringan besar dalam keterbatasan sumberdaya sebuah laptop. Dalam hal ini, perbandingan dengan penelitian yang diangkat adalah dari jenis prototipe yang akan dibangun, penelitian ini menekankan pada prototipe yang dibangun berdasarkan penggunaan switch (sebagai hardware), sehingga diharapkan terjawab pertanyaan apakah *OpenFlow* dapat berjalan tanpa menggunakan switch khusus untuk *OpenFlow*, sedang pada penelitian Lantz Bob, dkk, menekankan pada simulator mininet dalam skalabilitas yang virtual. Peneliti menggunakan mininet hanya sebagai pembanding pencapaian dari penelitian yang dilakukan. [6]

Prototipe dengan menggunakan *software open-source* yang tersedia dipasaran dengan deamon software pengembangan yang baru yang dibuat oleh Nascimento, dkk (2011). Prototipe yang dibuat adalah prototipe *RouteFlow server* yang berjalan pada OpenFlow switch dan kontroler SDN diletakkan terpisah, sehingga prototipe ini menggunakan 2 buah kontrol sekaligus dalam infrastrukturnya. [7]

Software-Defined Networking (SDN) atau split arsitektur menurut Shirazipour ,M., (2012) adalah sebuah konsep yang memungkinkan/ memperbolehkan operator jaringan untuk mengelola router dan switch secara fleksibel menggunakan *software* yang berjalan di server eksternal. Sedangkan ONF white Paper (2012), mendefinisikan

SDN adalah sebuah Sebuah arsitertur network baru dimana kontrol jaringan dipisahkan dari forwarding dan diprogram secara langsung. Gambar 2, menggambarkan bagaimana gambaran logical dari SDN arsitektur, yang merupakan jaringan pintar yang secara logikal tersentralisasi berdasarkan software (software-base). Selain itu ONF menyatakan bahwa dengan SDN tidak lagi membutuhkan protokol standar, tetapi cukup hanya menerima intruksi dari sebuah SDN kontroler.[8][9]



Gambar 1 SDN Arsitektur (sumber: ONF White Paper)

Mateo, M. (2009) mengatakan, Switch *OpenFlow* terdiri dari dua jenis, yang pertama adalah *hardware-base switch*, yang telah dijual secara komersial oleh beberapa *vendor*. Switch jenis ini telah memodifikasi *hardware*-nya, menggunakan TCAM dan menggunakan OS khusus untuk mengimplementasikan *Flow-Table* dan *OpenFlow* protokol. Jenis yang kedua adalah *software-base switch* yang menggunakan sistem UNIX / Linux untuk mengimplementasikan seluruh fungsi *OpenFlow* switch.[10]

Menurut McKeown, dkk., (2008) bahwa switch *OpenFlow* paling tidak memiliki 3 bagian yaitu (1) Sebuah tabel *flow* yang terhubung dengan sebuah aksi untuk setiap *flow* yang masuk. (2) Sebuah *Secure channel* yang mengkoneksikan antar switch dengan remote/eksternal proses (yang disebut kontroler). (3) Protokol *OpenFlow*, yang melakukan komunikasi antara switch dengan kontroler. [11]

Terdapat beberapa perbedaan mendasar dari switch komersial (switch biasa) dengan switch *OpenFlow*. Perbedaan tersebut menurut Chung Yik,EE., (2012), terlihat seperti pada tabel dibawah ini. [4]

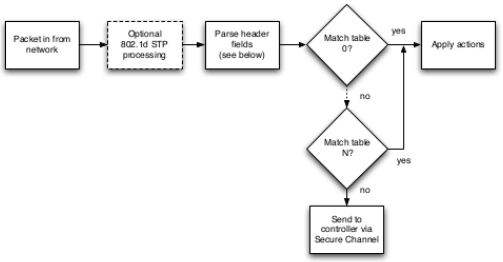
Tabel 1. Perbedaan switch *OpenFlow* dengan switch komersial (switch biasa) sumber: Chung Yik, EE., 2012.

Switch *OpenFlow*	**Switch komersial (switch biasa)**
Terpisahnya *control path* dan *data path*	*Control path* dan *data path* terletak pada perangkat yang sama (tidak terpisah)
Memungkinkan terjadi inovasi dalam jaringan	Membatasi inovasi dalam jaringan

 ISSN: 1411-3201

Menyediakan platform yang dapat diteliti dan diujicobakan pada jaringan sesungguhnya.	Tetap dan sulit untuk diujicobakan (dibuat tetap oleh *vendor*)
Fungsi yang dapat didefiniskan oleh user (dapat diprogram)	Arsitektur tertutup sehingga tidak dapat diprogram ulang
Setiap keputusan untuk melakukan pengiriman dilakukan oleh kontroler	Mengirimkan semua paket yang diterima keluar dari switch

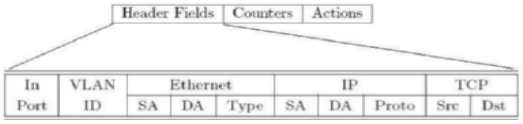
ONF (2009) menggambarkan proses pencocokan paket yang masuk kedalam switch seperti langkah pada gambar 3 dibawah ini.



Gambar 2 Pencocokan paket (sumber: ONF, 2009)

Paket data yang masuk kedalam switch akan dicek *field header* nya dan dicocokkan dengan tabel *flow*, bila cocok, paket tersebut akan diteruskan sesuai dengan *action* atau instruksi apa yang harus dilakukan dengan paket tersebut, namun bila tidak cocok pada tabel, akan dicocokkan lagi pada baris berikutnya hingga ke-n baris, bila hingga akhir dari tabel tidak ada kecocokan, maka paket akan dikirim ke kontroler menggunkan secure channel dan membiarkan kontroler mengolah paket tersebut. Pencocokan paket akan dilakukan berdasarkan prioritas dari paket tersebut, paket yang tidak memiliki *wildcat* (biasanya di beri notasi '*') atau terdefinisi secara spesifik akan diprioritaskan. [12]

McKeown, N., dkk (2008) menyatakan bahwa Protokol *OpenFlow* adalah sebuah standard terbuka yang memungkinkan peneliti melakukan kontrol langsung pada jalannya paket data yang akan di routing kan pada jaringan. *OpenFlow* melakukan sentralisasi terhadap kerumitan dari jaringan kedalam sebuah *software* kontroler, sehingga seorang administrator dapat mengaturnya dengan mudah, hanya dengan mengatur kontroler tersebut. McKeown, N., memperkenalkan konsep *OpenFlow* ini dengan ide awal adalah menjadikan sebuah network dapat di program/ dikontrol. [11]



Gambar 3 Open Flow Table fields (sumber : Mateo M. P., 2009)

Pada gambar 3, terlihat bahwa protokol *OpenFlow* terdiri dari 3 fields yaitu *Header Fields, Counter* dan *Action.* Mateo, M. P., (2009) menjelaskan, *header fields* adalah sebuah *packet header* yang mendefinisikan *flow, fields* nya terdiri dari enkapsulasi seperti enkapsulasi segmen pada protokol VLAN ethernet standard, *Counter* adalah sebuah *fields* yang menjaga jejak jumlah paket dan *byte* untuk setiap *flow*, dan waktu jejak paket terakhir cocok dengan *flow* (untuk membantu dalam membuang atau me-nonaktif-kan *flow*), dan *Action* adalah *fields* yang mendefinisikan bagaimana paket data akan diproses. [10]

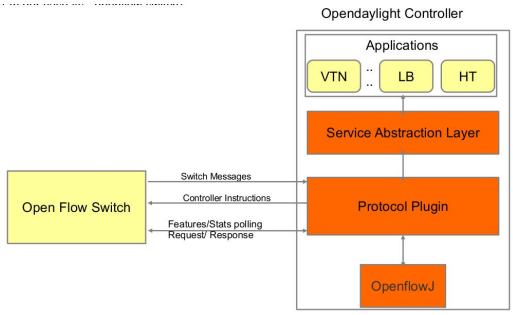
Menurut McKeown, N. (2008) Sebuah kontroler *OpenFlow* bertanggung jawab untuk menambahkan atau menghilangkan isi dari *flow* dari *OpenFlow* table yang ada didalam perangkat *OpenFlow* itu sendiri.

Ada 2 tipe dari kontroler yaitu kontroler statis, dapat berupa perangkat yang dapat menambahkan atau menghilangkan *flow* dari *flow* table secara statis. Kontroler dinamis, secara dinamis memanipulasi isi dari *flow* sehingga cocok untuk beberapa konfigurasi. [11]

Tanggung jawab kontroler menurut Vishnoi, A., dkk,(2013), dan tergambarkan pada gambar 4 adalah

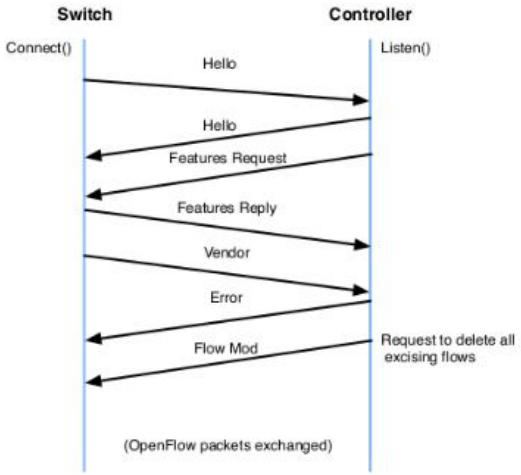
1. Menyediakan mekanisme untuk koneksi dan interaksi dengan *underlyingplatform* (dalam hal ini terhadap *OpenFlow* switch)
2. Menginterpretasikan pesan yang dikirim oleh switch *OpenFlow.*
3. Menyediakan semua instruksi pada setiap spesifikasi (baik spesifikasi yang dibutuhkan, tambahan atau keduanya) untuk dapat diprogram kedalam switch.
4. Memberikan mekanisme kepada switch untuk mendapatkan informasi/pendapat secara umum hingga ke yang detail dan harus dapat menginterpretasikan respon yang tepat. [13]

 ISSN: 1411-3201



Gambar 4 Tanggungjawab kontroler (sumber: Vishnoi,A., dkk, 2013)

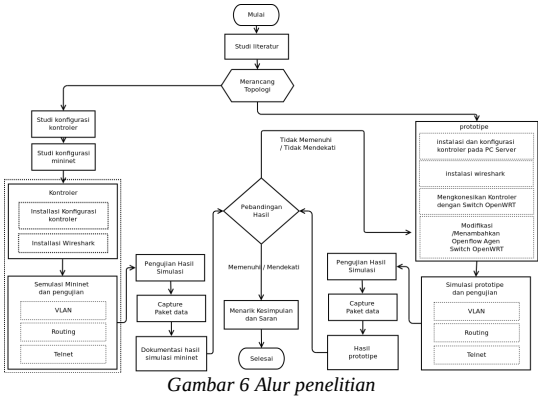
Appleman, M., dkk (2012), ketika *switch* terhubung dengan sebuah kontroler, yang pertama dilakukan kontroler adalah mengirimkan *Hello packet*, kemudian *switch* mengirimkan *Hello packet* Kembali, selanjutnya kontroler akan mengirimkan sebuah *Features Request packet*, *switch* harus membalas dengan sebuah *Features Reply*. Kemudian *switch* mengirimkan sebuah pesan *Vendor*, jika pesan *Vendor* tidak dimengerti oleh kontroler maka kontroler akan mengirimkan sebuah *Error packet* ke *switch*. Kontroler juga mengirimkan sebuah *Flow Mod packet* untuk menghapus semua *flow* dari *switch*. Proses lengkap dapat dilihat pada gambar 5 [2]



Gambar 5 Paket flow antara switch dan kontroler ketika switch terhubung (sumber : Appleman, M., dkk, 2012)

Metode Penelitian

Berikut adalah gambar diagram alir langkah-langkah penelitian yang akan dilakukan. Berdasarkan gambar 6 alur penelitian terdapat beberapa langkah, berikut adalah penjelasan dari tiap-tiap langkah tersebut



Gambar 6 Alur penelitian

1. Studi literatur, mengumpulkan bahan atau materi penelitian yang berupa literatur/referensi yang berhubungan dengan penelitian yang akan dilak- sanakan.
2. Studi konfigurasi kontroler, mempelajari konfigurasi *software* kontroler yang akan digunakan, baik cara konfigurasi, spesifikasi dan *troubleshoot* nya. Pada penelitian ini, *software* kontroler yang digunakan adalah *Opendaylight.*
3. Studi konfigurasi mininet, mempelajari konfigurasi dengan mininet *switch* simulator yang akan digunakan sebagai pembanding.
4. Merancang topologi untuk digunakan pada simulasi mininet dan prototipe/model yang nanti akan dibangun.
5. Melakukan simulasi dengan mininet, simulasi menggunakan 2 buah *switch* dan 4 buah *host* dan 1 buah kontroler. Kontroler diletakkan terpisah dari PC yang menjalani mininet, kontroler dijalankan pada *host* PC, sedangkan aplikasi VBox digunakan untuk menjalankan mininet simulator.
6. Variabel yang diteliti pada simulasi mininet adalah VLAN, routing dan telnet.
7. Pengujian, dilakukan pengujian dengan mengirimkan paket ICMP ke masing-masih *host*, sebelum di lakukan kontrol dengan kontroler, sesudah dipasangkan kontroler, dan sesudah dijalankan *OpenFlow* protokol.
8. Melakukan *capture data packet* menggunakan wireshark dengan wireshark yang diinstal langsung pada PC server kontroler.
9. Mendokumentasikan hasil yang diperoleh dari simulator mininet, untuk dijadikan acuan pada pengujian prototipe/model.
10. Membuat prototipe dengan menggunakan topologi yang sama dengan topologi yang digunakan pada simulator mininet.
11. Kontroler dikoneksikan pada *switch* OpenWrt pada *port* yang sama seperti pada simulator mininet.
12. Koneksi *port* prototipe dikoneksikan pada *port* yang sama seperti pada simulator mininet.

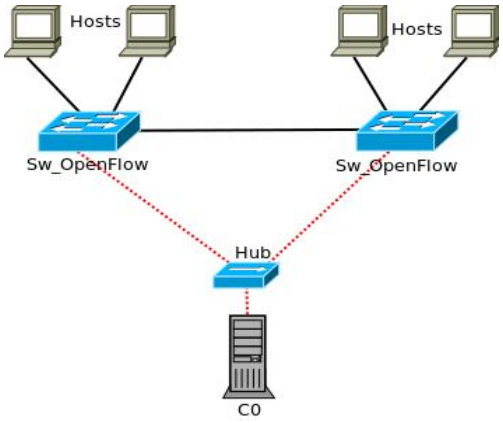
 ISSN: 1411-3201

13. instalasi kontrol dengan *Opendaylight*, pada PC/Laptop.
14. Melakukan konfigurasi pada kontroler dengan konfigurasi yang sama pada kontroler simulator mininet.
15. Pengujian pada prototipe sama dengan pengujian yang dilakukan pada simulator mininet dengan variabel yang sama.
16. Membandingkan hasil pengujian pada simulator mininet dan prototipe.
17. Bila *switch* OpenWrt belum memberikan hasil yang memenuhi / mendekati hasil mininet, maka akan dilakukan modifikasi pada *switch* OpenWrt, berupa penambahan/perubahan *OpenFlow* agen *switch* OpenWrt .
18. Mendokumentasikan hasil penelitian
19. Menarik kesimpulan dan saran.

Hasil dan Pembahasan

Penelitian ini secara umum terdiri dari 3 langkah utama, yaitu simulasi menggunakan simulator mininet sebagai pembanding untuk prototipe, pembuatan prototipe, dan membandingkan hasil simulasi mininet dengan hasil prototipe. Sebelum melakukan simulasi mininet, terlebih dahulu menentukan/ merancang topologi yang nantinya akan digunakan.

Topologi yang digunakan pada penelitian ini dapat dilihat pada gambar 7 berikut,



Gambar 7 Rancangan topologi

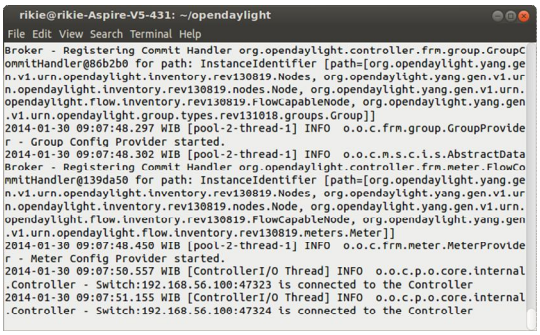
topologi ini akan digunakan pada simulasi dengan mininet dan pada prototipe.

Kontroler yang digunakan pada penelitian ini adalah kontroler Opendaylight versi 0.1. Kontroler Opendaylight telah mendukung *OpenFlow* 1.0 dan *OpenFlow* 1.3, dan Opendaylight dapat dijalankan pada semua *flatform* sistem operasi.

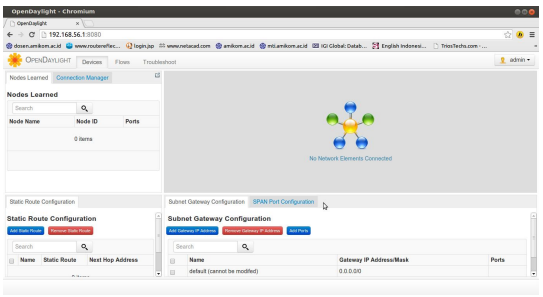
Pada penelitian ini, Opendaylight diinstal pada Laptop dengan spesifikasi sebagai berikut :

- CPU : Intel® Pentium(R) CPU 987 @1.50GHz × 2
- RAM : SODIMM DDR3 2Gb 1333MHz
- Hardisk : 500 Gbyte
- NIC : Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller (rev 0a)
- Operation System : Linux UBUNTU 12.04 LTS kernel 3.5.0-45-generic.

Pada gambar 8 adalah kontroler opendaylight yang berjalanpada terminal dan gambar 9 adalah GUI (*Grafic User Interface*) Opendaylight yang berjalan pada *web browser*. User dan password pada aplikasi ini secara *default* adalah “admin”, namun dapat dirubah dan ditambahkan dengan mudah sesuai dengan kebutuhan.



Gambar 8 Opendaylight pada terminal



Gambar 9 Opendaylight pada web browser

Mininet di instal pada kernel linux, dalam hal ini distro ubuntu, *image* simulator mininet ini telah tersedia pada situs *www.OpenFlow.org*, pada *image* ini, mininet diinstal pada ubuntu 13.04 dengan kernel *image* 3.8.0-15 generic. Setelah dijalankan, *image* ini menggunakan “mininet” sebagai *user* dan *password* .

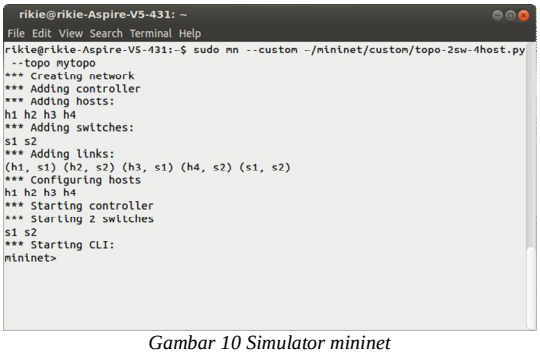
Topologi yang telah dirancang kemudian diimplementasikan pada mininet, sebelumbya dibuatkan coding dengan bahasa python yang dapat di eksekusi oleh mininet, adapun list coding untuk topologi yang telah dirancang adalah sebagai berikut

```
"""
Topologi prototipe
```

 ISSN: 1411-3201

```
Dua switch terkoneksi langsung, dengan masing-masing
switch terkoneksi dengan 2 host:
  host --- switch --- switch --- host
             |              |
           host           host
"""
from mininet.topo import Topo
class MyTopo( Topo ):
    "Topologi prototipe."
  def __init__( self ):

       "Membuat topologi."
      # Initialize topology
       Topo.__init__( self )

      # Add hosts and switches
   left0Host = self.addHost( 'h1' )
   right0Host = self.addHost( 'h2' )
   left1Host = self.addHost( 'h3' )
   right1Host = self.addHost( 'h4' )
   leftSwitch = self.addSwitch( 's1' )
   rightSwitch = self.addSwitch( 's2' )

      # Add links
self.addLink( left0Host,leftSwitch )
self.addLink( left1Host,leftSwitch )
self.addLink( leftSwitch,rightSwitch )
self.addLink( rightSwitch,right0Host )
self.addLink( rightSwitch,right1Host )

topos = { 'mytopo': ( lambda: MyTopo() )}
```

Setelah topologi dijalankan, akan menampilakan hasil seperti pada gambar 10 dibawah ini. Tanda "mininet> " menandakan bahwa simulator telah siap untuk digunakan.



Gambar 10 Simulator mininet

Untuk melihat *port* yang terkoneksi pada perangkat, dapat dilihat dengan perintah "*net*" pada CLI mininet dan akan tergambarkan bagaimana perangkat terkoneksi satu sama lainnya.

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth2
h3 h3-eth0:s1-eth2
h4 h4-eth0:s2-eth3
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h3-eth0 s1-eth3:s2-eth1
s2 lo:  s2-eth1:s1-eth3 s2-eth2:h2-eth0 s2-eth3:h4-eth0
c0
```

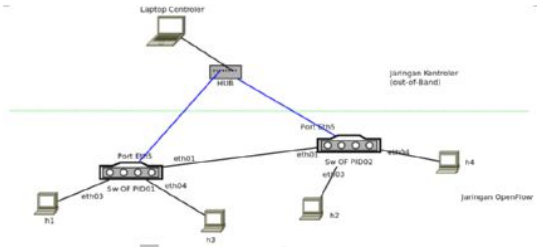
Karena pada prototipe, *port* ethernet pada PC penguji, menggunakan NIC dengan kecepatan maksimal 100Mb/s, maka pada simulasi mininet juga menggunakan besar *bandwidth* sebesar 100Mb/s dengan perintah *"--link tc,bw=100"*, dan *Ipbase* yang digunakan pada simulasi ini adalah 172.16.1.0/24.

Switch yang digunakan dalam prototipe adalah switch TP-LINK WR1043nd versi 1.11 dengan perubahan pada firmware yang digunakan. Firmware yang digunakan pada prototipe adalah firmware OpenWRT yang telah ditambahkan denngan agen *OpenFlow.*

Gambar 11 adalah proses *flashing* ROM pada switch prototipe dengan menggunakan *image firmware* yang telah ditambahkan agen *OpenFlow.*

Gambar 6 Image firmware diunggah ke ROM

Setelah switch OpenWrt-*OpenFlow* disiapkan, kedua switch tersebut diimplementasikan pada topologi yang telah dirancang sebelumnya. Koneksi antar switch dan *host* dapat dilihat pada gambar 12 dibawah ini.



Gambar 12 Koneksi antara host, switch dan kontroler

Konfigurasi IP yang digunakan adalah sebagai berikut, sedang kontroler diberikan *IP address* 192.168.1.76/24 seperti pada tabel 2 dibawah ini

Tabel 2. Konfigurasi IP dan *port* switch

Switch	IP address	Port Ke kontroler	Port terkoneksi
Switch (PID01)	192.168.1.11/24	Eth0.5	Eth0.1, eth0.3, eth0.4
Switch (PID02)	192.168.1.12/24	Eth0.5	Eth0.1, eth0.3, eth0.4

 ISSN: 1411-3201

Pemberian IP pada setiap switch dikonfigurasikan langsung pada file /etc/config/*OpenFlow* seperti dibawah ini, untuk switch 1 dan 2 dilakukan hal yang sama dengan PID yang berbeda.

```
config 'ofswitch'
    option 'dp' 'dp0'
    option 'dpid' '000000000001'
    option 'ofports' 'eth0.1 eth0.2 eth0.3
eth0.4'
option'ofctl''tcp:192.168.1.76:6633'
option 'mode' 'outofband'
```

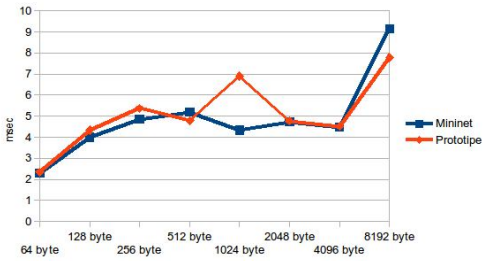
Sedang untuk IP dilakukan konfigurasi pada /etc/config/network dengan konfigurasi sebagai berikut:

```
...
config 'interface'
    option 'ifname' 'eth0.5'
    option 'proto' 'static'
    option 'ipaddr' '192.168.1.11'
    option 'netmask' '255.255.255.0'
```

Dari pengujian yang telah dilakukan dan dari data-data yang telah didapatkan, dapat dilakukan perbandingan antara hasil yang diperoleh dari mininet dan hasil yang diperoleh dari prototipe. Hasil uji yang dilakukan pada prototipe maupun mininet diambil nilai rata-rata, adapun perbandingan hasil uji tersebut adalah sebagai berikut.

Tabel 3. Hasil pengujian pengiriman paket ICMP

	(dalam msec)		
	Mininet	Prototipe	Gap
64 byte	2,288	2,341	0,053
128 byte	3,999	4,337	0,338
256 byte	4,855	5,376	0,521
512 byte	5,191	4,787	0,404
1024 byte	4,341	6,903	2,562
2048 byte	4,726	4,772	0,046
4096 byte	4,477	4,515	0,038
8192 byte	9,149	7,776	1,373
		Gap rata-rata	0,667

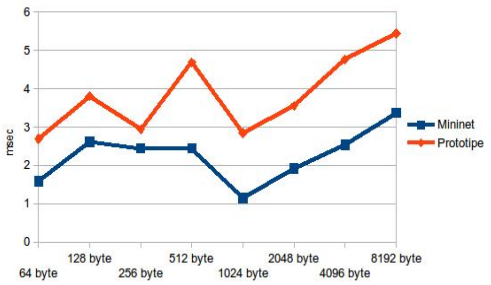


Gambar 13 Perbandingan Uji pengiriman paket ICMP

Pada pengujian prototipe dengan besar paket ICMP yang ditampilkan pada tabel 3 dan gambar 13 diatas menunjukkan nilai pada besar paket 1024byte menunjukkan angka 6.903msec. Hal ini dapat diakibatkan perbedaan respon switch 1 atau switch 2 terhadap perintah dari kontroler. Waktu yang tinggi pada paket ICMP 1024byte dapat pula disebabkan karena kestabilan *software* kontroler yang digunakan belum baik dan masih menggunakan versi awal. Hingga laporan ini ditulis,
kontroler opendaylight yang digunakan masih terus mengalami perbaikan.

Tabel 4. Hasil pengujian pengiriman paket ICMP-VLAN

	(dalam msec)		
	Mininet	Prototipe	Gap
64 byte	1,587	2,693	1,106
128 byte	2,61	3,807	1,197
256 byte	2,444	2,937	0,493
512 byte	2,441	4,687	2,246
1024 byte	1,149	2,844	1,695
2048 byte	1,920	3,556	1,636
4096 byte	2,540	4,762	2,222
8192 byte	3,371	5,435	2,064
		Gap rata-rata	1,582



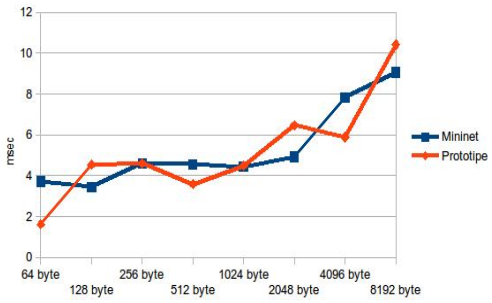
Gambar 14 Perbandingan Uji pengiriman paket ICMP-VLAN

Pengujian pengiriman paket ICMP-VLAN, memiliki selisih yang lebih besar dari pengujian pertama, dengan nilai rata-rata selisih sebesar 1.582 seperti terlihat pada tabel 4. Pada gambar 14 terlihat data prototipe berada diatas data mininet, hal ini disebabkan karena pada mininet, proses switch mendapatkan informasi dan memproses *flow* table dari kontroler jauh lebih cepat dibandingkan dengan prototipe, dikarenakan mininet menggunakan sumberdaya CPU laptop yang menjadi satu dengan kontroler, sedang pada prototipe, switch memproses paket dengan sumberdaya CPU yang terpisah dengan kontroler.

Tabel 5. Perbandingan Uji pengiriman paket ICMP - Routing

 ISSN: 1411-3201

		(dalam msec)	
	Mininet	Prototipe	Gap
64 byte	3,713	1,622	2,091
128 byte	3,449	4,545	1,096
256 byte	4,613	4,613	0,000
512 byte	4,56	3,585	0,975
1024 byte	4,437	4,499	0,062
2048 byte	4,94	6,480	1,540
4096 byte	7,842	5,890	1,952
8192 byte	9,069	10,421	1,352
		Gap rata-rata	1,134



Gambar 75 Perbandingan uji pengiriman paket ICMP - Routing

Data prototipe, seperti terlihat pada tabel 5 dan gambar 15, cendrung tidak stabil. Hal ini dapat disebabkan oleh *software* kontroler yang tidak stabil. Nilai pada pengujian prototipe dengan besar paket ICMP-Routing 2048byte menunjukkan angka 6.480msec, namun pada besar paket 4096byte cendrung turun hingga 5.890msec..

Pengujian telnet dilakukan dengan memberikan intruksi drop port 23 pada kontroler, sama halnya dengan simulator mininet, prototipe dapat melakukan pelarangan terhadap koneksi ke port 23 dan tidak ada perbedaan diantara keduanya.

Pengujian respon switch terhadap kontroler memperoleh hasil seperti pada tabel 6 dibawah ini.

Tabel 6. Nilai rata-rata pengujian respon

	Rata-rata	
	Sequance error	Packet Loss (%)
Mininet	4	33
Prototipe	4	44
Gap	0	11

Pada pengujian respon, terdapat selisih *packet loss* sebesar 11%, Lambannya respon yang diberikan oleh prototipe sehingga mengalami *packet loss* sebesar 44% dapat juga dipengaruhi oleh lambannya *initial hand-shake* yang terjadi antara kontroler dengan switch prototipe..

Kesimpulan dan Saran

Berdasarkan hasil pembahasan dan pengujian dari prototipe *software-defined network* dengan menggunakan protokol *OpenFlow* menggunakan ubuntu sebagai kontroler, maka dapat disimpulkan sebagai berikut:

1. Penelitian ini dapat memberikan pengetahuan yang lebih baik tentang infrastruktur *software-defined network* dimana dengan infrastruktur ini dapat mengembangkan jaringan dengan cepat dan tidak tergantung dengan salah satu *vendor*.
2. Prototipe yang dibangun telah berhasil dan mendekati hasil yang diperoleh simulator mininet, sehingga dapat dibangun sebuah infrastruktur *software-defined Network* dengan biaya yang relatif murah.
3. Penelitian membuktikan bahwa *OpenFlow* dapat berjalan pada switch biasa, seperti terlihat pada hasil yang disampaikan pada uji pengiriman paket ICMP dengan selisih 0.667, uji pengiriman paket ICMP - VLAN dengan selisih 1.582, dan dengan uji pengiriman paket ICMP - Routing dengan selisih 1.134 sedangkan pada uji telnet simulator mininet dan prototipe sama-sama dapat merespon pelarangan telnet yang diberikan pada kontroler. Pengujian respon switch terhadap kontroler memberikan hasil bahwa prototipe lebih lamban memberikan respon kembalinya kontroler dari kondisi *off* dengan besar kehilangan paket rata-rata sebesar 44%
4. Pengaruh kontroler sangat besar terhadap hasil yang dicapai oleh infrastruktur s*oftware-defined network*. karena kontrolerlah yang mengatur dan melakukan perintah terhadap switch yang terlibat dalam infrastruktur. Kestabilan prototipe juga berpengaruh terhadap kestabilan kecepatan pengiriman paket dari satu *host* ke *host* lainnya sehingga perlu untuk dikembangkan lagi.
5. Untuk mengimplementasikan prototipe ini, ada beberapa keterbatasan diantaranya adalah jumlah port yang sedikit, sehingga bila diimplentasikan pada jaringan dengan user yang banyak, switch yang digunakan switch juga banyak dan tentunya akan memberikan pengaruh terhadap performa dari infrastruktur. Tingkat kestabilan switch harus lebih ditingkatkan. Penggunaan hub untuk menghubungkan antara jaringan *OpenFlow* dan jaringan kontroler dapat berpengaruh seiring bertambahnya jumlah switch yang digunakan.

Prototipe pada penelitian ini sangat mungkin untuk ditingkatkan dan dikembangkan lebih lanjut, adapun beberapa saran dari peneliti adalah:

1. Pengembangan dengan menggunakan switch openWRT dengan tipe lain sehingga dapat melihat tingkat kestabilan pada tipe switch openWRT yang lain.
2. Pengujian dilakukan menggunakan kontroler opendaylight, sehingga masih bisa saja

 ISSN: 1411-3201

dikembangkan dengan menggunakan kontroler lain. Hasil yang berbeda bisa saja terjadi, dibandingkan dengan apa yang telah diperoleh pada penelitian ini.

3. Untuk mengimplementasikan prototipe ini, dapat digunakan kontroler yang lebih stabil. Dapat pula diujikan dengan menggunakan kontroler yang bersifat desentralisasi.

Daftar Pustaka

[1] Anonim, 10 Mei 2013, "Software-Defined Networking: The New Norm for Networks ", Open Networking Foundation. 2012, https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[2] Applemen,M., De Boer,M., 30 Mei 2013, "Performance Analysis of *OpenFlow* Hardware ", University Of Amsterdam 2012, http://staff.science.uva.nl/~delaat/rp/2011-2012/p18/report.pdf

[3] Naous, J., Erickson, D., Covington, G., A., Appenzeller, G., McKeown, N., 4 Mei 2013, "Implementing an *OpenFlow* Switch on the NetFPGA platform", ANCS '08, November 6–7, 2008, San Jose, CA, USA. Copyright 2008 ACM 978-1-60558-346-4/08/0011,www.yuba.stanford.edu/~jnaous/papers/ancs-*OpenFlow*-08.pdf

[4] Chung Yik,EE.,10 Mei 2013,"IMPLEMENTATION OF AN OPEN FLOW SWITCH ON NETFPGA" ,Universiti Teknologi Malaysia 2012, http://portal.fke.utm.my/fkelibrary/files/eechungyik/2012/15_EECHUNGYIK2012.pdf

[5] Kamus Besar Bahasa Indonesia, 3 November 2013, Pusat Bahasa Depertemen Pendidikan Nasional, 2008.

[6] Lantz,B., Heller,B., McKeown,N., 3 Mei 2013, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks" Monterey, CA, USA. Copyright 2010 ACM 978-1-4503-0409-2/10/10, http://klamath.stanford.edu/~nickm/papers/a19-lantz.pdf

[7] Nascimento,Marcelo R., Rothenberg,Christian E., Salvador,Marcos R., Magalh˜ es,Maur´cio F., Corr^ a,Carlos N.A., De Lucena, Sidney C., 16 Agustus 2013, Workshop de Pesquisa Experimental da Internet do Futuro , WPEIF 2011, http://sbrc2011.facom.ufms.br/files/workshops/wpeif/wpeif.pdf

[8] Shirazipour,M., John,W.,Kempf,J., Green,H.,Tatipamula,M., 3 Mei 2013, "Realizing Packet-Optical Integration with SDN and OpenFlow 1.1 Extensions", Communications (ICC), 2012 IEEE International Conference on ISSN:1550-3607, http://www.fp7-sparc.eu/assets/publications/15-SDN12_Packet-Opto-Integration-with-OF11.pdf

[9] Anonim, 10 Mei 2013, "Software-Defined Networking: The New Norm for Networks ", Open Networking Foundation. 2012, https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[10] Mateo,P. Manuel, 19 Mei 2013, "OpenFlow Switching Performance ", POLITECNICO DI TORINO 2009, http://www.OpenFlow.org/downloads/technicalreports/MS_Thesis_Polito_2009_Manuel_Palacin_OpenFlow.pdf

[11] McKeown,N., Anderson,T., Balakrishnan ,H., Parulkar,G., Peterson,L., Rexford ,J., Shenker ,S., Turner ,J., 1 Mei 2013, "OpenFlow: Enabling Innovation in Campus Networks" , Stanford University 2008, www.OpenFlow.org/documents/OpenFlow-wp-latest.pdf

[12] Anonim, 10 Mei 2013, "OpenFlow Switch Specification Version 1.0.0 (Wire Protocol 0x01) " Open Networking Foundation, 2009, www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/OpenFlow/OpenFlow-spec-v1.0.0.pdf

[13] Vinshoi, Anilkumar., Khumbhare, Abhijit., 21 Desember 2013, "Open Flow 1.3.1 Support: Controller View ", IBM, https://wiki.opendaylight.org/images/d/dc/OpenFlow1.3_Support_for_Opendaylight.pdf

Biodata Penulis

Rikie Kartadie, memperoleh gelar Sarjana Teknik (S.T) Universitas Pembangunan Nasional "Veteran" Yogyakarta, lulus tahun 2001. Saat ini sebagai Mahasiswa Magister Teknik Informatika STMIK AMIKOM Yogyakarta.

Ema Utami, memperoleh gelar Sarjana Sains (S.Si), Program Studi Ilmu Komputer FMIPA UGM, lulus tahun 1997. Tahun 2002 memperoleh gelar Magister Komputer (M.Kom) dari Program Ilmu Komputer UGM. Program Doktor pada Ilmu Komputer UGM, lulus tahun 2010. Saat ini sebagai Staf Pengajar program Magister Teknik Informatika STMIK AMIKOM Yogyakarta.

Eko Pramono, memperoleh gelar Sarjana Sains (S.Si), Fakultas MIPA jurusan Fisika Program Studi Elektronika dan Instrumentasi, lulus tahun 1995. Tahun 2006 memperoleh gelar Magister Teknik (M.T) dari Fakultas Teknik jurusan Teknik Elektro UGM. Saat ini sebagai Staf Pengajar program Magister Teknik Informatika STMIK AMIKOM Yogyakarta.