

## **BAB 3**

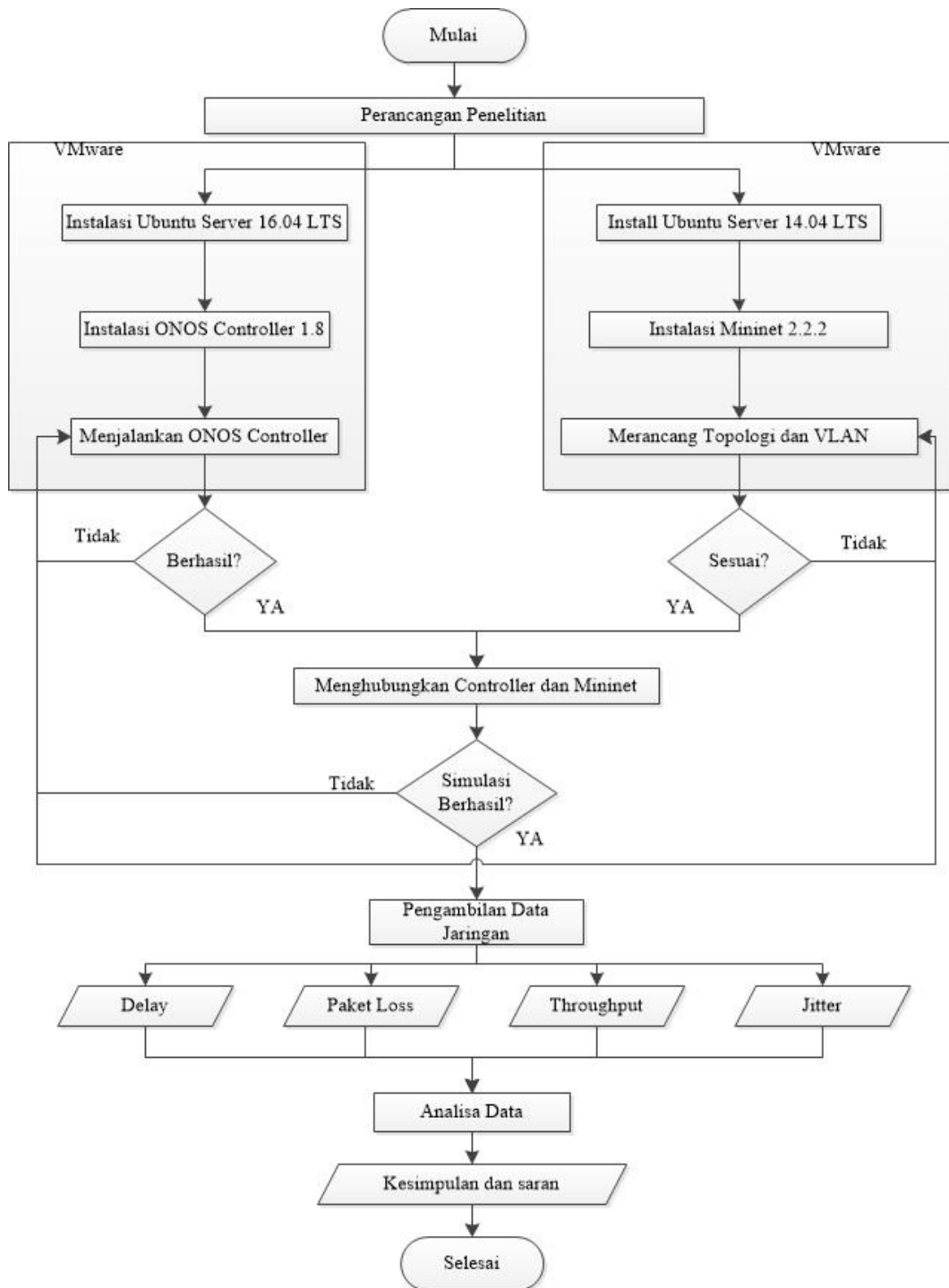
### **METODE PENELITIAN**

Pada bab ini dibahas mengenai proses perancangan dan implementasi analisis performansi VPLS (*virtual private LAN Service*) dengan Vlan *encapsulation* berbasis SDN (*software defined network*) menggunakan ONOS *controller*. Pengiriman paket menggunakan D-ITG dengan *protocol* UDP dari PC 1 ke PC 3. Dalam penelitian skripsi ini penulis menggunakan dua skenario, yaitu skenario pertama dengan *bandwidth* sebesar 10 *Mbps* pengiriman paket menggunakan D-ITG dari PC 1 ke PC 3, sedangkan PC 2 dan PC 4 dijadikan sebagai beban trafik menggunakan iPref dengan lima varian berbeda yaitu 2 *Mbps*, 4 *Mbps*, 6 *Mbps*, dan 8 *Mbps*, Nantinya akan mengirim paket secara bersamaan dan dilakukan 30 kali masing-masing percobaan.

Skenario kedua masih sama dengan skenario pertama tetapi *bandwidth* tidak di beri batas atau *unlimited*. Dengan demikian akan diketahui bagaimana kinerja sebuah jaringan SDN dengan *bandwidth* yang berbeda berdasarkan parameter *delay*, *jitter*, *throughput* dan *packet loss*.

#### **3.1 FLOWCHART PENELITIAN**

Pada gambar 3.1 menampilkan *flowchart* penelitian skripsi analisis performansi (VPLS) *virtual private LAN Service* dengan Vlan *encapsulation* berbasis (SDN) *software defined network* menggunakan ONOS *controller*.



**Gambar 3.1** *Flowchart* Penelitian.

### 3.2 TEMPAT DAN WAKTU PENGAMBILAN DATA

Pada proses pengerjaan skripsi ini penulis melakukan penelitian, perancangan dan pengujian jaringan di laboratorium komputer Institut Teknologi Telkom Purwokerto. Dalam pengerjaan skripsi ini penulis membutuhkan waktu dua bulan dalam perancangan hingga pengujian jaringan.

### 3.3 PERANGKAT PENELITIAN

#### 3.3.1 PERANGKAT KERAS (*Hardware*)

Dalam penelitian ini penulis menggunakan satu unit perangkat keras dengan spesifikasi sebagaimana ditampilkan pada table 3.1.

**Tabel 3.1 Spesifikasi Personal Komputer.**

Spesifikasi	Laptop
Merk	Lenovo
<i>Processor</i>	<i>Intel Core i5 @ 1.60 GHz</i>
<i>Memory (RAM)</i>	8 GB
<i>Operating System</i>	<i>Windows 8 Enterprise 64-bit</i>
VGA	<i>NVIDIA GT 720M 2 GB</i>

#### 3.3.2 PERANGKAT LUNAK (*Software*)

##### 1. *Mininet*

Mininet untuk membangun simulasi jaringan Vlan, jaringan dalam simulasi ini di buat secara *Virtual* dengan menggunakan 6 unit *switch* dan 4 unit *host* yang di beri *IP address* dan Vlan setiap *host*nya. *mininet* berjalan pada Sistem operasi *linux* 14.04 LTS. Table 3.2 menjelaskan *minimum system requiment* untuk menjalankan *software mininet 2.2.2*.

**Table 3.2 Minimum System Requirement Mininet 2.2.2.**

Spesifikasi	<i>Minimum requirements</i>
<i>Ubuntu</i>	10.04++
<i>Processor</i>	1 Core CPU
RAM	1024 MB/1 GB
<i>Harddisk</i>	10 GB

## 2. ONOS Controller

Digunakan sebagai *controller* yang nanti akan menjalankan fungsi *switch* dalam jaringan dan menampilkan sebuah *topologi* yang digunakan di *mininet*. Serta mengatur apa saja yang di butuhkan pada sebuah jaringan VPLS. Sistem operasi yang digunakan pada penelitian ini *linux server* 16.04 LTS. Tabel 3.3 menunjukan *system requiment* atau standarisasi perangkat ONOS yang di butuhkan agar dapat dijalankan secara maksimal.

**Tabel 3.3 System Requirements ONOS 1.8.**

Spesifikasi	<i>System requirements</i>
<i>Ubuntu</i>	16.04 LTS
<i>Processor</i>	2 Core CPU
<i>RAM</i>	2 GB
<i>Harddisk</i>	10 GB
<i>NIC</i>	1

## 3. VMware

Digunakan sebagai simulator instalasi *ubuntu Server* 16.04 LTS digunakan sebagai *controller* dan *ubuntu Server* 14.04 LTS sebagai *Mininet 2.2.2*. Berfungsi sebagai penghubung antara *controller layer* dan *physical layer*.

## 4. UBUNTU

*Ubuntu Server* 16.04 LTS di gunakan sebagai *controller* ONOS. Nantinya sebagai tempat dimana *controller* digunakan untuk mengatur sebuah jaringan SDN. *Mininet* juga menggunakan sistem operasi *ubuntu* versi 14.04 LTS digunakan sebagai simulasi perangkat jaringan SDN.

## 5. *Wireshark*

Digunakan dalam proses pengambilan data simulasi jaringan. Aplikasi *wireshark* men-*capture* informasi semua paket yang keluar masuk di sisi *client* saat dilakukan pengiriman paket data.

## 6. *PuTTY*

Digunakan untuk menghubungkan system operasi *ubuntu* menggunakan *SSH* agar lebih mudah saat meng-configuration perintah. Serta sebagai penghubung *host* pada jaringan SDN. *Putty* berjalan pada system operasi *windows* dapat di *download* secara gratis.

## 7. *Xming*

Digunakan untuk membuka CLI pada setiap *host* yang terhubung pada jaringan SDN. Nantinya *Xming* akan terhubung dengan pada *putty* menampilkan CLI pada *host*, sehingga dapat menulis perintah mengirim paket data antar *host*.

## 8. D-ITG

Digunakan untuk mengirimkan paket data dari sisi penerima ke pengirim, berupa *protocol* UDP.

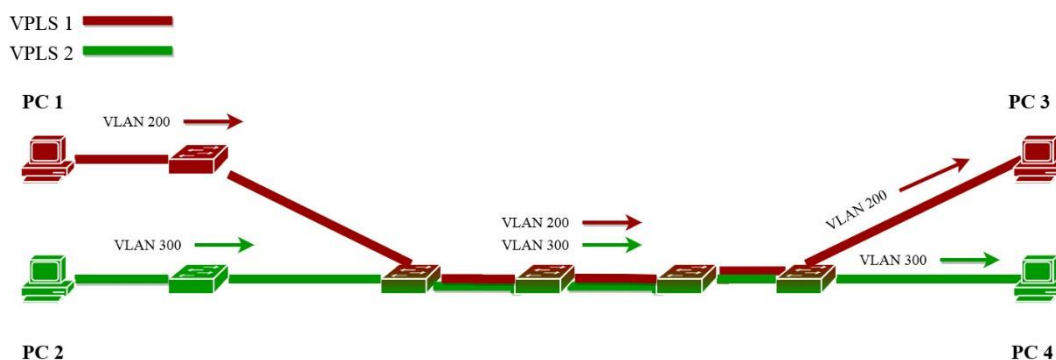
## 9. *iPerf*

Digunakan untuk memberikan beban trafik pada *bandwidth* di sebuah jaringan.

## 3.4 SIMULASI PENELITIAN

### 3.4.1 PERANCANGAN TOPOLOGI

Dalam penelitian ini dilakukan pengujian terdapat 4 *client* di ujung jaringan topologi dengan Vlan ID yang berbeda. Jaringan ini menggunakan alamat *IPv4* yang sudah tersimpan pada *mininet*. Dalam topologi jaringan dibuat dengan menggunakan 6 unit *switch* saling terhubung. Nantinya akan terdapat jalur VPLS yang akan di lalui *traffic*. Terlihat gambar 3.2 jaringan topologi VPLS dimana nantinya terdapat dua jalur yaitu VPLS1 dan VPLS2.



**Gambar 3.2 Topologi Jaringan VPLS.**

Pada gambar diatas menampilkan jaringan topologi *bus (linier)*, Pada bagian komputer PC1 menggunakan alamat *network* 192.168.10.1/32 dengan Vlan ID 200 dan pada wilayah salah satunya lagi pada computer PC3 menggunakan alamat *network* 192.168.10.3/32 dengan Vlan ID 200. Nantinya kedua buah *client* tersebut akan saling terhubung melalui VPLS1 jalur berwarna merah. Pada Komputer PC2 menggunakan alamat *network* 192.168.10.2/32 dengan Vlan ID 300 dan pada computer satunya lagi PC4 menggunakan alamat *network* 192.168.10.4/32 dengan Vlan ID 300.

Topologi jaringan VPLS menggunakan *OpenFlow* 1.0 dan *VSwitch* pada semua *switch* yang terdapat dalam jaringan. Berdasarkan topologi jaringan yang di tampilkan pada gambar 3.2 berikut ini rincian konfigurasi *IP* tiap *device* di tampilkan pada tabel 3.3.

**Tabel 3.4 Konfigurasi Device Jaringan VPLS.**

<i>Device</i>	<i>Interfaces</i>	<i>Mac</i>	<i>IP</i>	Vlan	VPLS
PC1	h1-eth0	00:00:00:00:01	192.168.10.1	200	VPLS1
PC2	h2-eth0	00:00:00:00:02	192.168.10.2	300	VPLS2
PC3	h3-eth0	00:00:00:00:03	192.168.10.3	200	VPLS1
PC4	h4-eth0	00:00:00:00:04	192.168.10.4	300	VPLS2

Gambar 3.2 jaringan VPLS dibuat pada *software Mininet* menggunakan program *python* yang di simpan pada folder *custom*. Gambar 3.3 *script python* yang digunakan untuk membuat topologi pada *software mininet*. *Self.add host* sendiri untuk menambahkan *host*, memberi *IP address* dan Vlan ID. *Self.add switch*

menambahkan berapa jumlah *switch* yang akan di butuhkan pada jaringan ini. Serta *self.add link* untuk menghubungkan perangkat.

```
        return r

hosts = { 'vlan': VLANHost }

class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):

        # Initialize topology
        Topo.__init__( self )

        # Menambah dan Memberi IP serta Vlan
        h1=self.addHost( 'h1', ip='192.168.1.1', cls=VLANHost, vlan=200)
        h2=self.addHost( 'h2', ip='192.168.1.2', cls=VLANHost, vlan=300)
        h3=self.addHost( 'h3', ip='192.168.1.3', cls=VLANHost, vlan=200)
        h4=self.addHost( 'h4', ip='192.168.1.4', cls=VLANHost, vlan=300)

        # Menambah Switch

        s1 = self.addSwitch( 's1' )
        s2 = self.addSwitch( 's2' )
        s3 = self.addSwitch( 's3' )
        s4 = self.addSwitch( 's4' )
        s5 = self.addSwitch( 's5' )
        s6 = self.addSwitch( 's6' )

        # Menhubungkan Host KeSwitch

        self.addLink(s1,h1)
        self.addLink(s2,h2)
        self.addLink(s6,h3)
        self.addLink(s6,h4)

        # Menghubungkan Antar Switch

        self.addLink(s1,s3)
        self.addLink(s2,s3)
        self.addLink(s3,s4)
        self.addLink(s4,s5)
        self.addLink(s5,s6)

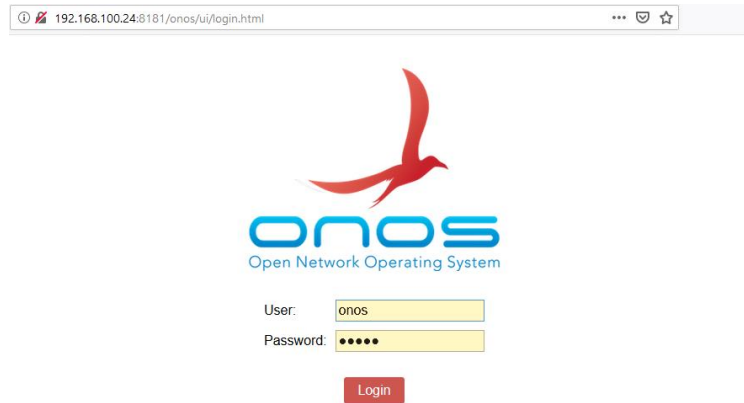
topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Gambar 3.3 *Script Python Mininet.*

### 3.4.2 PERANCANGAN *CONTROLLER*

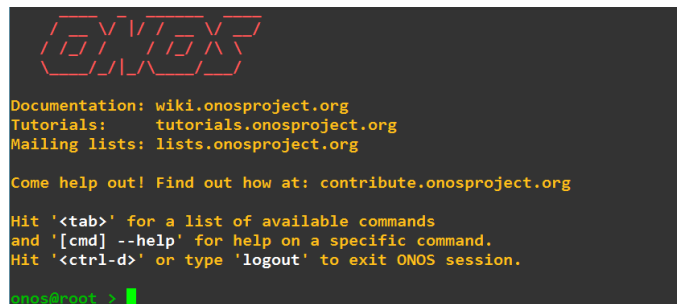
Perancangan *controller* ONOS pada jaringan berbasis SDN. Pada penelitian ini *controller* diinstall dalam sistem operasi *ubuntu Server 16.04 LTS*. Dalam instalasi ONOS di perlukan instalasi *Java8* dan *curl*. Setelah ONOS terinstall terdapat dua *interface* yang dapat di akses yaitu mode CLI dan GUI. Gambar 3.4

ONOS GUI dimana ONOS dapat di akses pada *browser* tetapi harus sudah terinstall SSH pada *ubuntu server* terlebih dahulu.



**Gambar 3.4 ONOS GUI**

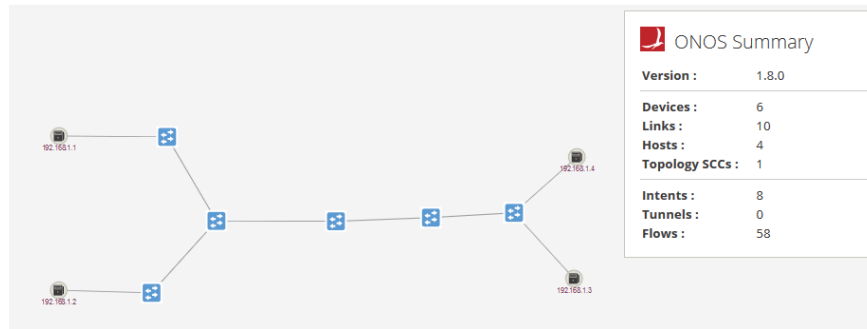
ONOS GUI dapat di akses menggunakan browser dengan alamat *IP address controller*. Gambar 3.5 ONOS CLI tampilan CLI ketika di jalankan pada system operasi *ubuntu server* menggunakan *putty*.



**Gambar 3.5 ONOS CLI.**

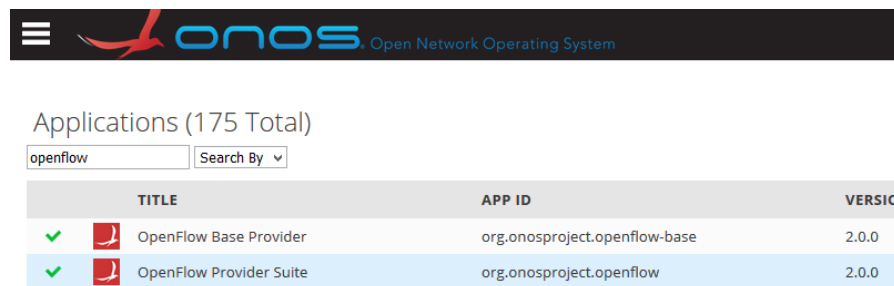
Dalam CLI digunakan untuk mengidentifikasi setiap *interface host* pada topologi di *mininet*. Gambar 3.6 topologi ONOS yang ditampilkan melalui *controller*.





**Gambar 3.6 Topologi ONOS.**

Pemasangan *OpenFlow* sangat penting karena dibutuhkan sebagai penghubung antara *controller* dan *mininet*. Sehingga *controller* ONOS dapat terhubung dengan *mininet* dan membaca semua *device* yang ada. Gambar 3.7 *OpenFlow* yang harus di instalasi.



**Gambar 3.7 Instalasi *OpenFlow*.**

Instalasi VPLS gambar 3.8 berfungsi agar dapat membuat VPLS yang akan menghubungkan antar Vlan yang nantinya di konfigurasi pada CLI.



**Gambar 3.8 Instalasi VPLS.**

Menghubungkan *controller* dengan *mininet* agar *device* dapat di baca oleh *controller* ONOS. Gambar 3.9 perintah untuk penghubung *mininet* dan *controller*.

```
mininet@mininet-vm:~/mininet/custom$ sudo mn --custom vpls.py --topo mytopo
--controller=remote,ip=192.168.100.24 --switch ovsk --mac
```

**Gambar 3.9 Menghubung Mininet dan Controller.**

Pada gambar 3.9 perintah “*sudo mn*” berfungsi untuk menjalankan program *mininet*, “*-- custom VPLS.py -- topo mytopo*” berfungsi untuk memanggil topologi yang telah di buat secara manual, “*-- controller = remote, IP=192.168.100.24*” berfungsi untuk menghubungkan *controller* dengan *IP address*, “*-- switch ovsk*” artinya topologi menggunakan *open vSwitch*, dan “*-- mac*” mengatur *mac address* secara *automatic*, *mac address* akan berurutan sehingga lebih mudah di kenali.

Jika sudah terhubung maka topologi telah dapat digunakan tetapi semua *host* belum terhubung dengan ONOS. sehingga jika melakukan *pingall* maka semua *host* tidak dapat terhubung. Gambar 3.10 Pengecekan *test ping* topologi.

```
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8
*** Adding links:
(s1, s2) (s2, s3) (s3, s4) (s4, s1) (s5, h1) (s5, s1) (s6, h2) (s6, s2) (s7, h3)
(s7, s3) (s8, h4) (s8, s4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 8 switches
s1 s2 s3 s4 s5 s6 s7 s8 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X
h2 -> X X X
h3 -> X X X
h4 -> X X X
*** Results: 100% dropped (0/12 received)
```

**Gambar 3.10 Pengecekan Test Ping Topologi.**

Identifikasi *host* kedalam *controller* ONOS, berfungsi agar *controller* ONOS dapat membaca semua *host* yang terdapat pada *mininet*. Gambar 3.11 *Host Interfaces* pada ONOS.

```
id=00:00:00:00:01/200, mac=00:00:00:00:00:01, locations=[of:0000000000000001], vlan=200, ip(s)=[192.168.1.1
], innerVlan=None, outerTPID=unknown, provider=of:org.onosproject.provider.host, configured=false
id=00:00:00:00:02/300, mac=00:00:00:00:00:02, locations=[of:0000000000000002], vlan=300, ip(s)=[192.168.1.2
], innerVlan=None, outerTPID=unknown, provider=of:org.onosproject.provider.host, configured=false
id=00:00:00:00:03/200, mac=00:00:00:00:00:03, locations=[of:0000000000000003], vlan=200, ip(s)=[192.168.1.3
], innerVlan=None, outerTPID=unknown, provider=of:org.onosproject.provider.host, configured=false
id=00:00:00:00:04/300, mac=00:00:00:00:00:04, locations=[of:0000000000000004], vlan=300, ip(s)=[192.168.1.4
], innerVlan=None, outerTPID=unknown, provider=of:org.onosproject.provider.host, configured=false
```

**Gambar 3.11 Hosts Interfaces.**

Pada gambar 3.11 *hosts interfaces* terdapat beberapa informasi *host* yang terhubung ke *controller*, seperti *id*, *mac*, *locations*, *vlan*, dan *IP address*.

*Host* akan masuk VPLS pada ONOS setelah semua *host* telah teridentifikasi oleh *Controller*. Memasukan setiap *host* pada masing-masing VPLS terlihat seperti gambar 3.12 VPLS1 dan VPLS2. Terlihat h1 dan h3 di dalam VPLS 1 yang memiliki ID Vlan yang sama yaitu 200, sedangkan h2 dan h4 didalam VPLS 2 yang memiliki ID Vlan yang sama juga yaitu 300.

```
onos> vpls show
Configured VPLSs
-----
VPLS name: VPLS1
Associated interfaces: [h1, h3]
Encapsulation: VLAN
-----
VPLS name: VPLS2
Associated interfaces: [h2, h4]
Encapsulation: VLAN
-----
onos>
```

**Gambar 3.12 VPLS1 dan VPLS2.**

Pengecekan secara menyeluruh dengan cara *pingall* pada *mininet*. Jika sudah benar maka akan terlihat seperti gambar 3.13 cek *pingall*.

```
mininet>
mininet>
mininet>
mininet>
mininet> pingall
*** Ping: testing ping reachability
h1 -> X h3 X
h2 -> X X h4
h3 -> h1 X X
h4 -> X h2 X
*** Results: 66% dropped (4/12 received)
mininet>
```

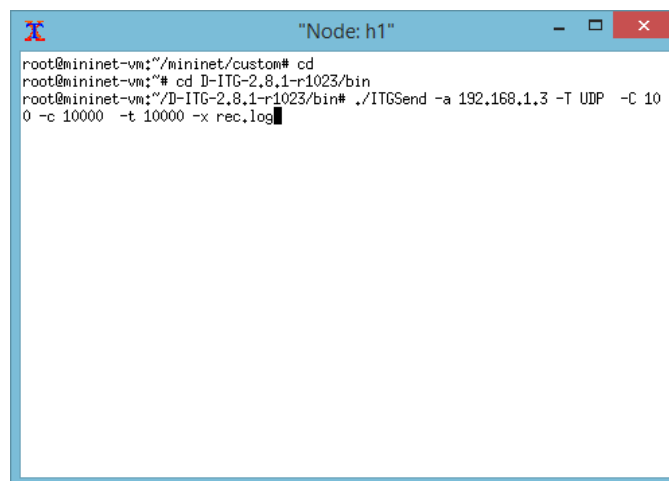
**Gambar 3.13 Cek Pingall.**

Gambar 3.13 cek *pingall* terlihat h1 hanya dapat mengirim ke h3 karena memiliki VPLS dan Vlan ID 200 yang sama. Jika memiliki VPLS yang berbeda maka *host* tidak dapat saling terhubung walaupun ID Vlannya sama.

### 3.5 PENGUKURAN *DELAY*

*Delay* adalah waktu yang dibutuhkan data untuk menempuh jarak dari *node* satu ke *node* lainnya asal. *Delay* dapat dipengaruhi oleh jarak, *delay* yang bagus jika makin kecil jarak antara pengirim dan penerima maka dapat memperkecil waktu pengiriman. *Delay* juga dapat berpengaruh dari media fisik atau juga waktu proses yang lama.

SDN menggunakan *Xming* untuk memunculkan CLI *host* untuk melakukan pengiriman paket data dari satu *node* ke *node* lainnya. Gambar 3.14 pengiriman paket data melalui *console* dengan D-ITG dari sisi pengirim menggunakan *protocol* UDP dengan perintah “*./ITGSend -a 192.168.1.3 -T UDP -C 100 -c 10000 -t 10000 -x rec.log*”. Artinya *./ITGSend* adalah untuk mengirimkan paket, *-a* yaitu alamat ip address yang di tuju, *-T* jenis protocol yang dikirimkan berupa UDP, *-C* *rate* yaitu jumlah pengiriman paket perdetiknya sebanyak 100 paket , *-c* *size* yaitu ukuran data yang dikirimkan ke penerima. Kemudian *-t* *duration* adalah waktu durasi paket yang dibutuhkan 10 detik (10000 *ms*).



```
root@mininet-vn:~/mininet/custom# cd
root@mininet-vn:~/mininet/custom# cd D-ITG-2.8.1-r1023/bin
root@mininet-vn:~/mininet/custom/D-ITG-2.8.1-r1023/bin# ./ITGSend -a 192.168.1.3 -T UDP -C 100 -c 10000 -t 10000 -x rec.log
```

**Gambar 3.14 Pengujian Data Pengirim.**

Gambar 3.15 Pada sisi penerima dengan perintah “*./ITGRecv -l receiver.log*” artinya *./ITGRecv* pc tersebut digunakan sebagai penerima, dan *-l receiver.log* mengubah log menjadi *txt format*.

```

"Node: h3"
root@mininet-vn:~/mininet/custom# cd
root@mininet-vn:~/mininet/custom# cd D-ITG-2,8,1-r1023/bin
root@mininet-vn:~/mininet/custom/D-ITG-2,8,1-r1023/bin# ./ITGRecv -l receiver.log

```

**Gambar 3.15** Pengujian Data Penerima.

Melihat hasil dengan *start capture* menggunakan *wire shark*. *Wire shark* adalah sebuah *software* yang dapat menganalisa suatu jaringan yang terhubung. Dilihat gambar 3.16 hasil dari *ping test* untung melihatnya dengan mem-*filter* jenis *internet control message protocol (ICMP)*. *Time delta from previous displayed frame* adalah nilai waktu *delay* setiap *reply* dengan satuan *second*.

11181	5942.764846000	192.168.1.3	192.168.1.1	ICMP	1516
11182	5943.765507000	192.168.1.1	192.168.1.3	ICMP	1516
11183	5943.765542000	192.168.1.3	192.168.1.1	ICMP	1516
11184	5944.765381000	192.168.1.1	192.168.1.3	ICMP	1516
11185	5944.765390000	192.168.1.3	192.168.1.1	ICMP	1516
11188	5945.764964000	192.168.1.1	192.168.1.3	ICMP	1516
11189	5945.764998000	192.168.1.3	192.168.1.1	ICMP	1516
11190	5946.765695000	192.168.1.1	192.168.1.3	ICMP	1516
11191	5946.765744000	192.168.1.3	192.168.1.1	ICMP	1516

```

Interface id: 0
Encapsulation type: Ethernet (1)
Arrival Time: Oct 16, 2019 22:07:50.734894000 PDT
[Time shift for this packet: 0.000000000 seconds]
Epoch Time: 1571288870.734894000 seconds
[Time delta from previous captured frame: 0.000034000 seconds]
[Time delta from previous displayed frame: 0.000034000 seconds]

```

**Gambar 3.16** *Wireshark Capture*.

Melakukan pengecekan *wireshark* terlebih dahulu dan Jika memang sudah cocok ukuran yang di kirimkan *host* terlihat di *wireshark*, maka akan di lakukan pengecekan *delay*.

Pada percobaan data pertama dapat di ketahui gambar 3.17 pengiriman dari hasil setelah paket ter-*filter* UDP. Mengirimkan paket sebanyak 913 dengan total waktu pengiriman 9.996 detik dan kecepatan pengiriman sebesar 7.338 *Mbps*.

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	939	913	97.231%	0	0.000%
Between first and last packet	11.015 sec	9.996 sec			
Avg. packets/sec	85.250	91.338			
Avg. packet size	9765.955 bytes	10042.000 bytes			
Bytes	9170232	9168346	99.979%	0	0.000%
Avg. bytes/sec	832548.283	917213.683			
Avg. MBit/sec	6.660	7.338			

**Gambar 3.17 Hasil Pengujian Data Pengirim.**

Disisi penerima gambar 3.18 penerima dari hasil setelah paket ter-*filter* UDP. Paket di terima sebanyak 913 dengan total waktu penerima 9.995 detik dan kecepatan penerima sebesar 7.331 *Mbps*.

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	945	913	96.614%	0	0.000%
Between first and last packet	11.014 sec	9.995 sec			
Avg. packets/sec	85.797	91.344			
Avg. packet size	9704.165 bytes	10032.273 bytes			
Bytes	9170436	9159466	99.880%	0	0.000%
Avg. bytes/sec	832584.861	916392.528			
Avg. MBit/sec	6.661	7.331			

**Gambar 3.18 Hasil Pengujian Data Penerima.**

Hasil pengujian mendapatkan Perhitungan untuk parameter *delay* mendapatkan total waktu *delay* 9.995 *Second*. Paket yang berhasil diterima sebanyak 913 paket. Berikut ini persamaan rumus 2.2 untuk contoh perhitungan *delay*:

$$Delay = \frac{\text{Total waktu Delay}}{\text{Total paket yang di terima}}$$

$$Delay = \frac{9.995}{913} S$$

$$Delay = 0.010947426 S$$

$$Delay = 10.947 ms$$

Hasil 10.947ms dikategorikan pada tabel 3.6 adalah sangat bagus pada pengukuran *delay* standar versi *TIPHON*.

**Tabel 3.5 Standarisasi nilai *delay* versi *TIPHON TS 101 329-2*. [18]**

Kategori <i>Delay</i>	Besar <i>Delay</i>
Sangat bagus	$0 < 150 \text{ ms}$
Bagus	$150 \text{ ms} < 250 \text{ ms}$
Sedang	$250 \text{ ms} < 350 \text{ ms}$
Jelek	$350 < 450 \text{ ms}$

### 3.6 PENGUKURAN *PAKET LOSS*

*Packet Loss* adalah persentase hasil paket data yang tidak dapat terkirim hingga tujuannya. Karena disebabkan *corrupt* atau terjadinya gangguan pada pengiriman dari perangkat *switch*. Paket data yang di kirim sebanyak 913 paket disisi peneriman. Diterima sebanyak sebanyak 913 paket data. Terlihat tidak ada paket data yang gagal terkirim atau *corrupt* pada gambar 3.18, semua paket data telah terkirim ke sisi penerima. Berikut ini persamaan rumus 2.3 untuk contoh perhitungan *packet loss*:

$$Packet Loss = \frac{\text{Paket data dikirim} - \text{Paket data yang diterima}}{\text{Paket data yang dikirim}} \times 100\%$$

$$Packet Loss = \frac{913 - 913}{913} \times 100\%$$

$$Packet Loss = 0 \%$$

**Tabel 3.6 Standar *packet loss* versi *TIPHON TS 101 329-2*. [18]**

Kategori <i>Packet Loss</i>	<i>Packet Loss</i>
Sangat bagus	0 %
Baik	3 %
Cukup	15 %
Buruk	25 %

Hasilnya 0% dikategorikan pada tabel 3.7 adalah sangat bagus pada pengukuran *packet loss* standarisasi versi *TIPHON*.

### 3.7 PENGUKURAN *TROUGHPUT*

*Troughput* adalah kemampuan dalam pengiriman paket data dari jaringan tersebut. *Troughput* sendiri berbeda dengan *bandwidth*. Jika *bandwidth* itu wadahnya maka *troughput* kecepatan dalam wadah tersebut. Dapat di ketahui nilai *troughput* dari sisi penerima sebesar 7.331 *Mbps*.

### 3.8 PENGUKURAN *JITTER*

*Jitter* merupakan variasi *delay* pengiriman paket yang terjadi pada jaringan IP antara *source* dan *destination*. Besarnya nilai *jitter* yang dihasilkan dipengaruhi oleh variasi beban trafik dan besarnya tumbukan (*congestion*) antar paket pada jaringan IP. Berikut ini persamaan rumus 2.4 untuk contoh perhitungan *Jitter*:

$$Jitter = \frac{\text{Variasi Delay}}{\text{Total paket}}$$

$$Jitter = \frac{0.241563}{912}$$

$$Jitter = 0.0002648717 \text{ s}$$

$$Jitter = 0.2648 \text{ ms}$$

**Tabel 3.7 Standar *Jitter* berdasarkan *TIPHON TS 101 329-2*. [18]**

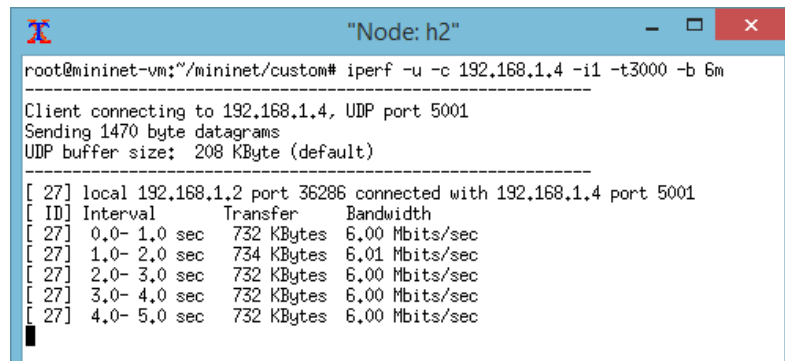
Kategori <i>Jitter</i>	Peak <i>Jitter</i>
Sangat bagus	0 <i>ms</i>
Bagus	75 <i>ms</i>
Sedang	125 <i>ms</i>
Jelek	225 <i>ms</i>

Hasilnya 0.2648 *ms* dikategorikan pada tabel 3.7 adalah sangat bagus pada pengukuran *Jitter* standarisasi versi *TIPHON*.



### 3.9 BEBAN TRAFIK

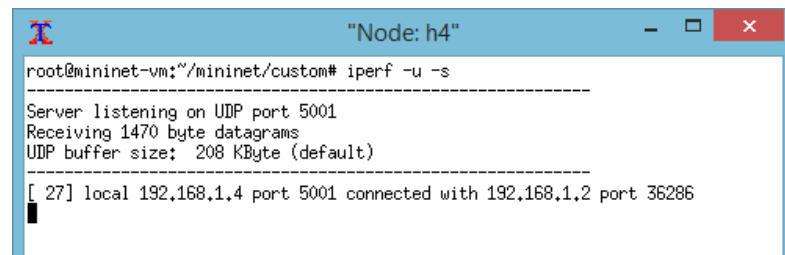
Memberikan beban trafik menggunakan *iPerf* dengan variasi yang berbeda - beda sebesar 2 *Mbps*, 4 *Mbps* 6 *Mbps* dan 8 *Mbps* dari pc 2 ke pc 4. Gambar 3.19 mengirimkan beban trafik dengan perintah “*iperf -u -c 192.168.1.4 -i1 -t3000 -b 6* “. -u artinya mengirimkan *protocol* UDP, -c mengirimkan alamat ip tujuan, -i interval waktu, -t lama waktu yang dibutuhkan dan -b besar beban trafik yang akan di kirimkan di sisi *server*.



```
root@mininet-vm:~/mininet/custom# iperf -u -c 192.168.1.4 -i1 -t3000 -b 6m
-----
Client connecting to 192.168.1.4, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 27] local 192.168.1.2 port 36286 connected with 192.168.1.4 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 27] 0.0- 1.0 sec   732 KBytes  6.00 Mbits/sec
[ 27] 1.0- 2.0 sec   734 KBytes  6.01 Mbits/sec
[ 27] 2.0- 3.0 sec   732 KBytes  6.00 Mbits/sec
[ 27] 3.0- 4.0 sec   732 KBytes  6.00 Mbits/sec
[ 27] 4.0- 5.0 sec   732 KBytes  6.00 Mbits/sec
```

**Gambar 3.19 Beban Trafik Pengirim.**

Gambar 3.20 Pada sisi penerima dengan perintah “*iperf -u -s*” -u artinya *protocol* yang digunakan berupa UDP dan -s sebagai *server* penerima paket dari *client*.



```
root@mininet-vm:~/mininet/custom# iperf -u -s
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 27] local 192.168.1.4 port 5001 connected with 192.168.1.2 port 36286
```

**Gambar 3.20 Beban Trafik Penerima**