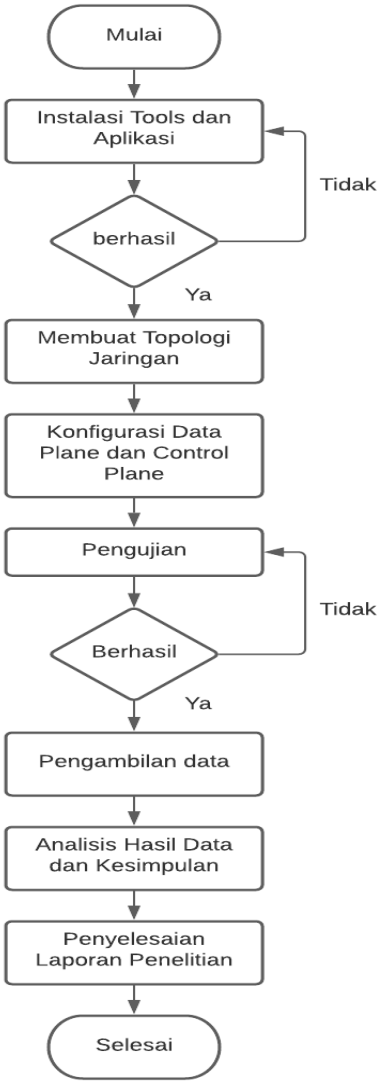
BAB III
METODOLOGI PENELITIAN

Secara garis besar simulasi penelitian ini ditujukan untuk mengimplementasikan *Software Defined Network* menggunakan *protocol routing Open shortest path first version 3* (OSPFv3). Selanjutnya Quality of Service (QoS) dari hasil tersebut dibandingkan dengan QoS dari jaringan konvensional menggunakan protokol routing OSPFv3. Diagram alur pada penelitian merupakan proses secara bertahap yang dilakukan untuk mendapatkan data kemudian dianalisis. Penelitian ini dilakukan dengan mengacu pada *flowchart* simulasi, seperti yang tertera pada Gambar 3.1.



Gambar 3.1 *FlowChart* Proses Simulasi

Pada Gambar 3.1 tahap pertama melakukan Instalasi Tool dan Aplikasi yang di butuhkan untuk melakukan sebuah simulasi pada perancagan *Software Defined Network*. Tahap berikutnya yaitu merancang topologi jaringan yang akan digunakan pada Mininet. Pada mininet pembuatan topologi memerlukan *script* topologi, dimana menggunakan bahasa pemrograman *python.* Dalam topologi jaringan itu terdapat perangkat *switch, Host* dan SDN *controller* yang jumlah dari setiap perangkat akan diuraikan pada pembahasan selanjutnya. *Controller* SDN yang digunakan adalah *POX controller*.

Tahap selanjutnya konfigurasi *data plane* dan *control plane*, *data plane* menjalankan emulator Mininet untuk konfigurasi *ip address* pada topologi yang sudah dibuat. Pada konfigurasi *control plane* berfungsi untuk menghubungkan antara topologi pada Mininet dengan *controller* yaitu POX. Hasil konfigurasi kemudian dilakukan cek konektifitas pada topologi dengan perintah ping pada Mininet. Jika terdapat kegagalan dalam cek konektifitas maka perlu dilakukan pengecekan ulang dari *data plane* dan *control plane* dan konfigurasi ulang jika diperlukan. Proses pengambilan data dilakukan setelah proses implementasi pada program selesai dilakukan atau dijalankan. Pengambilan data menggunakan perangkat lunak *Distributed Internet Traffic Generator (D-ITG),* Dalam hal ini D-ITG digunakan untuk komunikasi data komunikasi background *traffic*. Selanjutnya dari hasil trafik dilakukan analisis *jitter, throughput, delay,* dan *packet loss* yang didapat dari hasil menjalankan implementasi jaringan dari topologi dan *routing* yang digunakan.

3.1 Perangkat Simulasi

Penelitian ini dilakukan dalam bentuk simulasi untuk mendapatkan hasil datanya. Simulasi yang dilakukan membutuhkan beberapa alat pendukung agar dapat terlaksana. Alat pendukung yang digunakan berupa perangkat keras (*hardware*) dan perangkat lunak (*software*). Komponen utama tersebut diuraikan sebagai berikut :

3.1.1 Perangkat keras

Perangkat keras yaitu perangkat berbentuk fisik yang digunakan dalam penelitian. Penelitian ini menggunakan perangkat keras berupa satu unit PC yang menggunakan sistem operasi Windows 7 64-Bit dengan spesiifikasi seperti tabel 3.1. PC ini digunakan untuk menjalankan aplikasi emulator jaringan. Emulator jaringan yang digunakan adalah Mininet dan GNS3.

Tabel 3.1 Spesifikasi PC

Processor	AMD A12-9700P RADEON R7 @ 2.50 GHz
RAM	8,00 GB
Hardisk	1.000 GB
OS	Windows 10

3.1.2 Perangkat lunak

Perangkat lunak berupa perangkat yang tidak terlihat secara fisik. Perangkat lunak berupa perangkat yang terinstal pada sistem operasi Windows 7. Perangkat lunak yang digunakan untuk mensimulasikan penelitian ini sebagai berikut :

a. *VirtualBox*

   Pada penelitian ini *VirtualBox* menjalankan OS *linux Ubuntu* yang difungsikan untuk menjalankan Mininet yang terinstal *controller* POX. *VirtualBox* diinstal pada laptop yang menggunakan OS Windows 7 sehingga perlunya *VirtualBox* untuk menjalankan OS Linux Ubuntu.

b. *Mininet*

   Mininet adalah suatu software emulator yang memungkinkan untuk melakukan prototyping pada jaringan yang luas dengan hanya menggunakan satu mesin [15]. Simulasi jaringan dilakukan di dalam aplikasi mininet.

c. *MiniEdit*

   MiniEdit menggunakan bahasa pemprograman *python*. Miniedit adalah sebuah graphical user interface untuk perancangan skenario jaringan SDN [23]. *MiniEdit* terdapat didalam Mininet emulator yang digunakan untuk membuat *custom* topologi berbasis GUI pada *Mininet*.

d. POX *Controller*

POX adalah *platform* yang digunakan untuk pengembangan dan pemodelan pada *network control software*. POX menggunakan *python* dalam bahasa pemrogramannya [7].

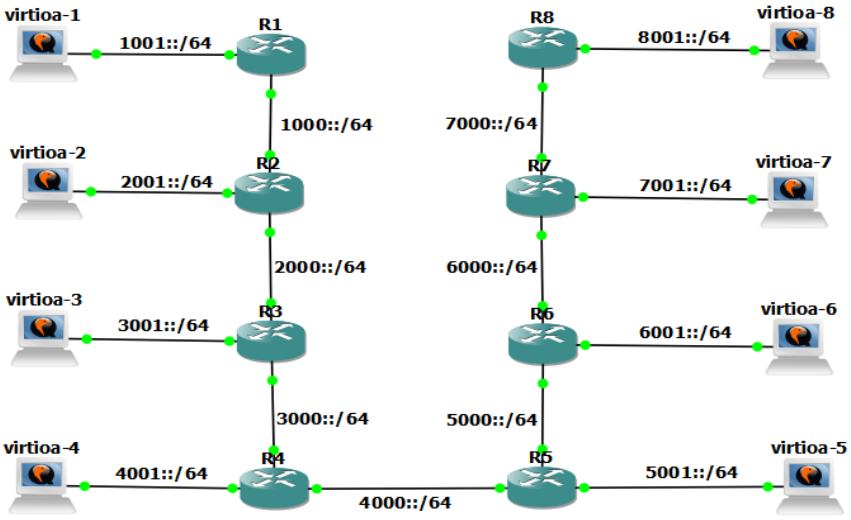
e. GNS3

GNS3 adalah sebuah program graphical network simulator yang dapat mensimulasikan topologi jaringan yang lebih kompleks dibandingkan dengan simulator lainnya. Program ini dapat dijalankan di berbagai sistem operasi, seperti Windows, Linux, atau MacOS X [24]. GNS3 digunakan sebagai emulator jaringan yang akan digunakan untuk mensimulasikan jaringan konvensional.

3.2 Perancangan Topologi Jaringan

Istilah topologi jaringan mengacu pada bagaimana bentuk perangkat jaringan yang terkoneksi apabila dilihat secara fisik (tampak kelihatan bentuk fisik jaringan) yang membentuk pola tertentu. Untuk saling terhubung perangkat dalam jaringan membutuhkan media transmisi kabel. Untuk transmisi kabel yang digunakan yaitu kabel UTP dan jenis topologi jaringan bus mengunakan 8 switch.

Dalam simulasi ini menggunakan dua simulator yaitu GNS3 untuk membuat jaringan konvensional dan emulator mininet yang terinstal pada ubuntu untuk membuat simulasi jaringan SDN *(Software Defined Network)*. Pada jaringan konvensional akan digunakan 8 perangkat *RouterOS Cisco* 7200 dan 8 *Host* yang disimulasikan dalam aplikasi GNS3, sedangakan untuk jaringan SDN menggunakan 1 *Controller*, 8 *switch OpenFlow* 1.3 dan 8 *Host* yang disimulasikan dalam emulator *Mininet*. Topologi jaringan menggunakan jaringan konvensional dapat dilihat pada Gambar 3.2.



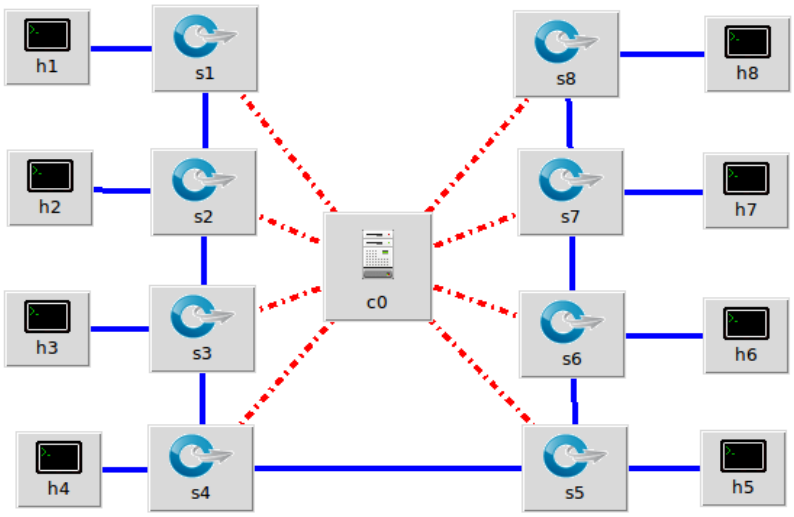
Gambar 3.2 Topologi jaringan pada GNS3

Pada gambar 3.2 dibuat dan disimulasikan pada aplikasi GNS3, router ditunjukkan dengan nama R dan *Host* dinamai dengan nama PC, topologi dapat diuraikan bahwa setiap *router* terhubung dengan masing-masing 1 *Host*, seperti *Router* 1 (R1) terhubung dengan *Host* 1 (PC1). Setiap *router* juga terhubung dengan *router* didekatnya seperti *Router* 1 (R1) terhubung dengan *Router* 2 (R2), *Router* 3 (R3) terhubung dengan *router* 2 (R2) dan *router 4* (R4). Pada tabel 3.2 diuraikan alamat ip yang ada pada setiap perangkat.

Tabel 3.2 Alamat IP Perangkat pada Jaringan Konvensional

Perangkat	**Network ID**	**Perangkat**	**Network ID**
Router 1	1000::/64	Router 8	7000::/64
	1001::/64		8001::/64
Router 2	1000::/64	Router 7	6000::/64
	2000::/64		7000::/64
	2001::/64		7001::/64
Router 3	2000::/64	Router 6	5000::/64
	3000::/64		6000::/64
	3001::/64		6001::/64
Router 4	3000::/64	Router 5	4000::/64
	4000::/64		5000::/64
	4001::/64		5001::/64
Host 1	1001::/64	Host 8	8001::/64
Host 2	2001::/64	Host 7	7001::/64
Host 3	3001::/64	Host 6	6001::/64
Host 4	4001::/64	Host 5	5001::/64

Topologi jaringan menggunakan jaringan *Software Define Network* (SDN) dapat dilihat pada Gambar 3.3.



Gambar 3.3 Topologi jaringan SDN pada mininet

Topologi pada gambar 3.3 setiap *switch OpenFlow* 1.3 dinamai dengan huruf S atau *Switch* 1 dengan nama (s1), perangkat *Host* dinamai dengan huruf h atau *Host* 1 dinamai menjadi (h1) dan perangkat *controller* dinamai menjadi (co). Setiap *switch* terhubung dengan masing-masing 1 *Host*, seperti s1 terhubung dengan h1 dan setiap *switch* terhubung dengan switch didekatnya seperti *switch* 1 (s1) terhubung dengan *switch* 2 (s2), *switch* 3 (s3) dan *switch* 4 (s4) begitu seterusnya sehingga membentuk topologi *Bus*. *Controller* terhubung langsung ke setiap switch, hal ini yang memungkinkan konfigurasi jaringan dilakukan secara terpusat.

Pengalamatan IP pada perangkat *Router/Switch* dan *Host* pada topologi jaringan SDN pada Gambar 3.3 diuraikan dalam Tabel 3.3 sebagai berikut.

Tabel 3.3 Alamat IP perangkat pada Jaringan SDN

Perangkat	**MAC**	**Perangkat**	**IP**
Switch 1	00:00:00:00:01:01	*Host* 1	10.0.0.1/24
Switch 2	00:00:00:00:01:02	*Host* 2	10.0.0.2/24
Switch 3	00:00:00:00:01:03	*Host* 3	10.0.0.3/24
Switch 4	00:00:00:00:01:04	*Host* 4	10.0.0.4/24
Switch 5	00:00:00:00:01:05	*Host* 5	10.0.0.5/24
Switch 6	00:00:00:00:01:06	*Host* 6	10.0.0.6/24
Switch 7	00:00:00:00:01:07	*Host* 7	10.0.0.7/24
Switch 8	00:00:00:00:01:08	*Host* 8	10.0.0.8/24
Controller	127.0.0.1	-	-

Perangkat Switch pada jaringan SDN sudah memiliki alamat IP *loopback* dan untuk pengaturan alamat IP pada MiniEdit untuk perangkat switch juga tidak tersedia, perangkat switch hanya dapat diatur mac addressnya.

3.3 Konfigurasi Jaringan

Pada bagian ini akan dijelaskan proses konfigurasi jaringan. Konfigurasi jaringan dilakukan pada simulasi topologi jaringan yang berbeda yaitu:

3.3.1 Konfigurasi pada jaringan konvensional

Dalam konfigurasi ini, proses dilakukan pada aplikasi GNS3. Disini kita mengatur alamat IP pada setiap perangkat *router* dan *Host*, selanjutnya menjalankan protokol *routing OSPFv3* pada perangkat *router cisco 7200*. Konfigurasi ini menggunakan 8 *router* dan menggunakan topologi bus. mengatur alamat IP pada setiap *interface* perangkat *router cisco 7200*. Setelah pengaturan alamat IP dilakukan pengaturan *protocol routing OSPFv3* pada masing- masing perangkat *router*.

Pada gambar 3.4 dilakukan konfigurasi IPv6 di Routing Protokol OSPFv3 pada *router* 1 (R1) dan masing-masing *router* dikonfigurasi seperti *router* 1.

```
R1#configure terminal
R1(config)#ipv6 unicast-routing
R1(config)#int fa0/0
R1(config-if)#ipv6 add 1000::1/64
R1(config-if)#no shutdown
R1(config-if)#exit
R1(config)#int fa1/0
R1(config-if)#ipv6 add 1001::1/128
R1(config-if)#no shutdown
```

Gambar 3.4 Konfigurasi alamat IP pada *router*

Setelah alamat IP pada masing-masing *interface* setiap *router* telah diatur, maka konfigurasi untuk *protocol routing OSPFv3* dapat dilakukan. Pada gambar 3.5 dilakukan konfigurasi *routing OSPFv3* pada *router* 1 (R1). Pengaturan *routing* dan konfigurasi dilakukan pada setiap *interface* pada masing-masing *router*

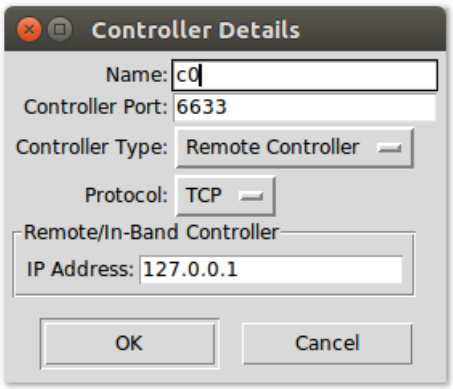
```
R1(config)# ipv6 router ospf 1
R1(config-router)#router-id 1.1.1.1
R1(config-router)#exit
R1(config)#int fa0/0
R1(config-if)#ipv6 ospf 1 area 0
R1(config-if)#exit
R1(config)#int fa1/0
R1(config-if)#ipv6 ospf 1 area 0
```

Gambar 3.5 Konfigurasi protokol *routing* OSPFv3

3.3.2 Konfigurasi pada jaringan SDN

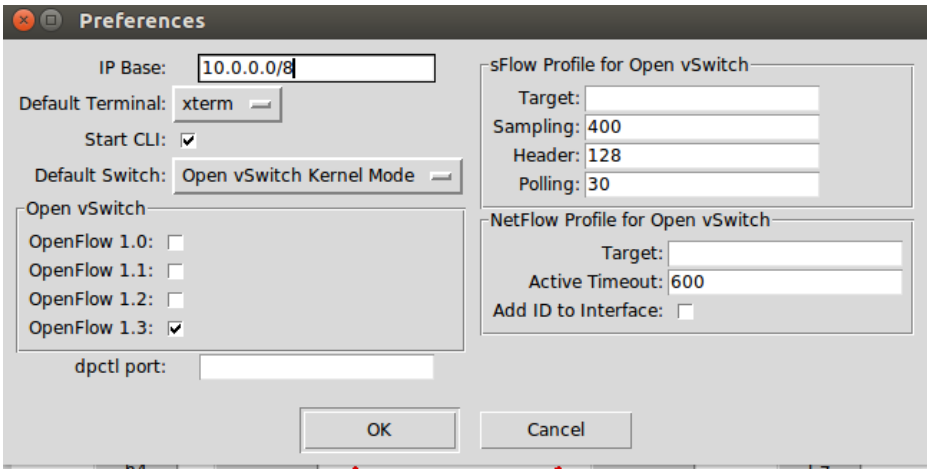
Pada konfigurasi ini, dilakukan pembuatan topologi dengan menggunakan aplikasi *MiniEdit* yang mana aplikasi tersebut terdapat dalam aplikasi *Mininet* yang diinstal di dalam sistem operasi Ubuntu. Topologi yang dibuat terdiri dari 1 *controller POX* yang terhubung ke 8 *switch OpenFlow* 1.3 yang terhubung dengan masing-masing 1 *Host*.

Pada gambar 3.6 merupakan konfigurasi *controller* yang dilakukan pada aplikasi *miniedit*. *Controller Type* yang awalnya *OpenFlow Reference* diubah menjadi *remote controller*. Pengubahan *controller* menjadi remote berfungsi agar *controller* dapat digunakan secara *remote* oleh *mininet* melalui IP 127.0.0.1 yang merupakan ip loopback.



Gambar 3.6 Konfigurasi *remote controller*

Setelah proses konfigurasi pada *controller* selesai dan topologi siap untuk di jalankan, selanjutnya tekan centang Start CLI pada menu Edit agar dapat mengakses *Host* ketika topologi pada MiniEdit dijalankan. Seperti yang ditunjukkan pada gambar 3.7.



Gambar 3.7 Setting *MiniEdit*

Selanjutnya konfigurasi topologi yang telah dibuat disimpan dan diexport menjadi 2 script yaitu script python dan script MiniEdit. Hal ini dilakukan agar mempermudah dalam pemanggilan atau menjalankan file topologi tersebut baik melalui MiniEdit maupun dengan *Command Line Interface* (CLI) pada mininet. Script Python ditunjukan pada Gambar 3.8.

```
def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                         controller=RemoteController,
                         ip='127.0.0.1',
                         protocol='tcp',
                         port=6633)

    info( '*** Add switches\n')
    s7 = net.addSwitch('s7', cls=OVSKernelSwitch)
    s3 = net.addSwitch('s3', cls=OVSKernelSwitch)
    s4 = net.addSwitch('s4', cls=OVSKernelSwitch)
    s5 = net.addSwitch('s5', cls=OVSKernelSwitch)
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch)
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
    s8 = net.addSwitch('s8', cls=OVSKernelSwitch)
    s6 = net.addSwitch('s6', cls=OVSKernelSwitch)
```

```
    info( '*** Add hosts\n')
    h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)
    h5 = net.addHost('h5', cls=Host, ip='10.0.0.5', defaultRoute=None)
    h3 = net.addHost('h3', cls=Host, ip='10.0.0.3', defaultRoute=None)
    h8 = net.addHost('h8', cls=Host, ip='10.0.0.8', defaultRoute=None)
    h6 = net.addHost('h6', cls=Host, ip='10.0.0.6', defaultRoute=None)
    h4 = net.addHost('h4', cls=Host, ip='10.0.0.4', defaultRoute=None)
    h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)
    h7 = net.addHost('h7', cls=Host, ip='10.0.0.7', defaultRoute=None)

    info( '*** Add links\n')
    net.addLink(s4, s5)
    net.addLink(s1, s2)
    net.addLink(s2, s3)
    net.addLink(s3, s4)
    net.addLink(s8, s7)
    net.addLink(s7, s6)
    net.addLink(s6, s5)
    net.addLink(h1, s1)
    net.addLink(h2, s2)
    net.addLink(h3, s3)
    net.addLink(h4, s4)
    net.addLink(s5, h5)
    net.addLink(s6, h6)
    net.addLink(s7, h7)
    net.addLink(s8, h8)


    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    info( '*** Starting switches\n')
    net.get('s7').start([c0])
    net.get('s3').start([c0])
    net.get('s4').start([c0])
    net.get('s5').start([c0])
    net.get('s2').start([c0])
    net.get('s1').start([c0])
    net.get('s8').start([c0])
    net.get('s6').start([c0])

    info( '*** Post configure switches and hosts\n')

    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```

Gambar 3.8 *Script Python Custom* Topologi pada *mininet*

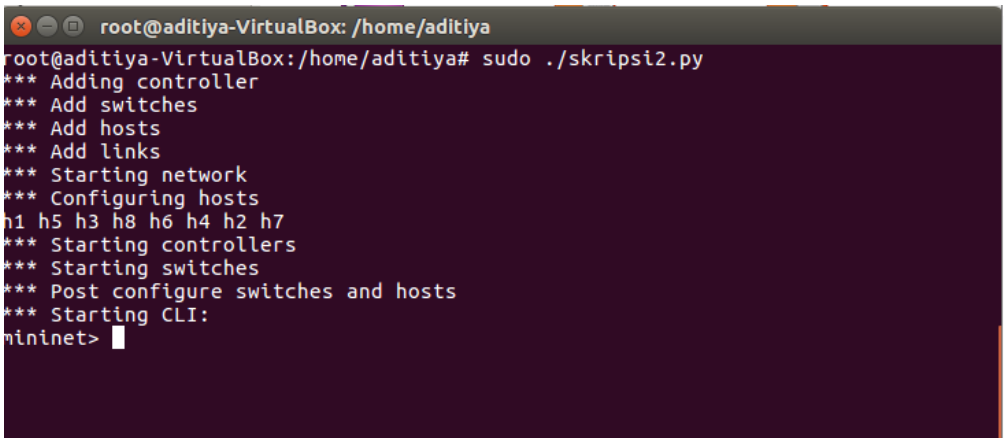
Pada *script* topologi ditunjukkan semua ip yang terdapat pada setiap perangkat dan link yang menghubungkan ke setiap perangkat. *Script routing* OSPF pada jaringan SDN dapat dilihat pada gambar 3.9.

```
# These next two imports are common POX convention
from pox.core import core
import pox.openflow.libopenflow_01 as of

# Even a simple usage of the logger is much nicer than print!
log = core.getLogger()

# This table maps (switch,MAC-addr) pairs to the port on 'switch' at
# which we last saw a packet *from* 'MAC-addr'.
# (In this case, we use a Connection object for the switch.)
table = {}

# To send out all ports, we can use either of the special ports
# OFPP_FLOOD or OFPP_ALL.  We'd like to just use OFPP_FLOOD,
# but it's not clear if all switches support this, so we make
# it selectable.
all_ports = of.OFPP_FLOOD

# Handle messages the switch has sent us because it has no
# matching rule.
def _handle_PacketIn (event):
  packet = event.parsed

  # Learn the source
  table[(event.connection,packet.src)] = event.port

  dst_port = table.get((event.connection,packet.dst))

  if dst_port is None:
    # We don't know where the destination is yet.  So, we'll just
    # send the packet out all ports (except the one it came in on!)
    # and hope the destination is out there somewhere. :)
    msg = of.ofp_packet_out(data = event.ofp)
    msg.actions.append(of.ofp_action_output(port = all_ports))
    event.connection.send(msg)
  else:

# Since we know the switch ports for both the source and dest
    # MACs, we can install rules for both directions.
    msg = of.ofp_flow_mod()
```

```
    msg.match.dl_dst = packet.src
    msg.match.dl_src = packet.dst
    msg.actions.append(of.ofp_action_output(port = event.port))
    event.connection.send(msg)

    # This is the packet that just came in -- we want to
    # install the rule and also resend the packet.
    msg = of.ofp_flow_mod()
    msg.data = event.ofp # Forward the incoming packet
    msg.match.dl_src = packet.src
    msg.match.dl_dst = packet.dst
    msg.actions.append(of.ofp_action_output(port = dst_port))
    event.connection.send(msg)

    log.debug("Installing %s <-> %s" % (packet.src, packet.dst))

def launch (disable_flood = False):
  global all_ports
  if disable_flood:
    all_ports = of.OFPP_ALL

  core.openflow.addListenerByName("PacketIn", _handle_PacketIn)

  log.info("Pair-Learning switch running.")
```

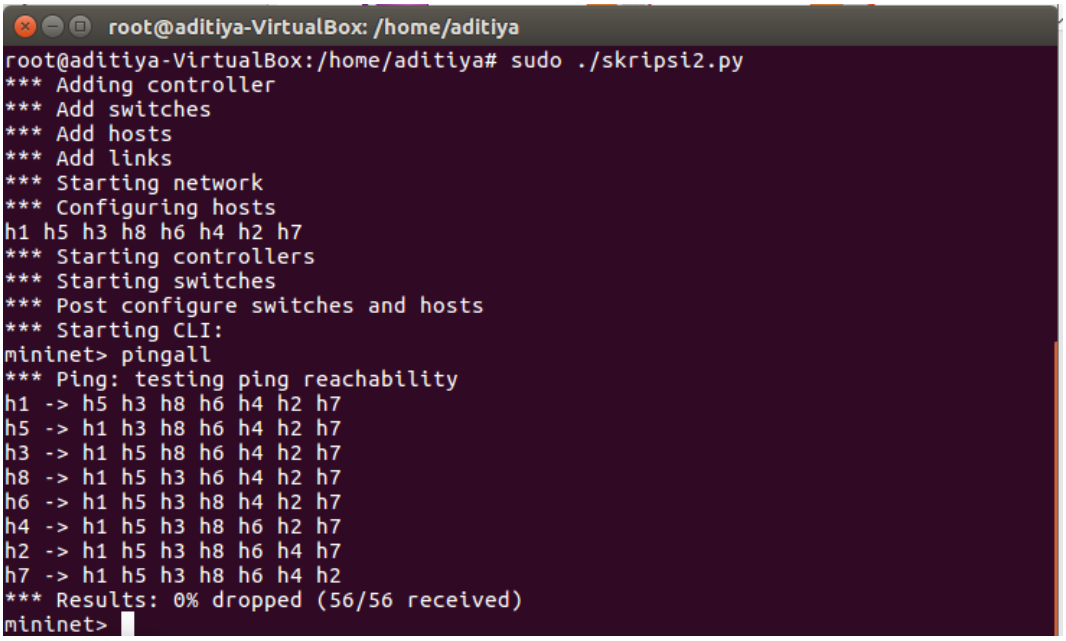
Gambar 3.9 *Script Algorit*ma *routing OSPFv3*

Pada Gambar 3.9 merupakan script python yang akan menjalankan algoritma *routing OSPFv3 m*enggunkan *controller pox.* Program tersebut disimpan dengan nama "l2_pairs.py" pada direktori *pox controller*, forwarding. Fungsi dari program tersebut untuk mengaktifkan POX controller yang telah menjalankan algoritma routing OSPFv3. Program harus diaktifkan atau dijalankan terlebih dahulu kemudian menjalankan *custom* topologi agar custom topologi dapat mendeteksi dan terkoneksi dengan *controller*. Setelah program diaktifkan dilakukan pemanggilan atau menjalankan topologi seperti yang terlihat pada gambar 3.10.



Gambar 3.10 Perintah menjalankan topologi

Menjalankan topologi dapat dilakukan dengan 2 cara yg bisa dilakukan yaitu dengan menggunakan perintah *Command Line Interface* (CLI) seperti pada gambar 3.10 dimana dilakukan pemanggilan terhadap file "./skripsi2.py" yang berisi custom topologi yang telah dibuat atau bisa juga melalui *MiniEdit* kemudian memanggil file custom topologi yang sudah dibuat dalam bentuk ".mn" dan menjalankannya sperti yang terlihat pada gambar 3.11.



Gambar 3.11 Pengujian topologi

Untuk menguji topologi yang sudah dibuat dilakukan pengecekan konektifitas dengan menjalankan perintah "pingall". Kemudian akan terlihat apakah setiap *Host* dapat terhubung satu sama lain atau tidak seperti yang terlihat pada gambar 3.11. Ketika semua *Host* dapat terhubung maka akan ditampilkan hasil "0% dropped".

3.4 Pengambilan Data

Terdapat 2 skenario pengambilan data yaitu pengambilan data pada jaringan konvensional dan jaringan SDN. Pada kedua skenario dilakukan hal yang sama yaitu dengan mengirimkan paket UDP dan TCP menggunakan aplikasi D-ITG dengan beban *traffic* 20 MB, 40 MB, 60 MB, 80 MB dan 100 MB sebanyak 30 kali. Pada D-ITG dilakukan konfigurasi *script* untuk mengahasilkan *traffic* yang dibutuhkan. Berikut berberapa parameter yang di isi :

1. -c : kode ini memperesentasikan besarnya data yang akan dikirim dalam 1 paket dengan satuan *byte*.
2. -C : kode ini mempresentasikan banyak nya paket yang akan dikirimkan.

3. -t : kode ini mempresentasikan lamanya paket akan dikirim dengan durasi yang sesuai dengan perintah ini, yaitu dengan satuan *millisecond.*

Dari penjelasan diatas, maka untuk mengirimkan ukuran data sebesar 7.5MB, 10 MB dan 12.5 MB digunakan konfigurasi sebagai berikut :

1. -c 34952 -C 30 -t 20000, untuk menghasilkan ukuran data sebesar 20 MB.
2. -c 69905 -C 30 -t 20000, untuk menghasilkan ukuran data sebesar 40 MB.
3. -c 104857 -C 30 -t 20000, untuk menghasilkan ukuran data sebesar 60 MB.
4. -c 139810 -C 30 -t 20000, untuk menghasilkan ukuran data sebesar 80 MB.
5. -c 174762 -C 30 -t 20000, untuk menghasilkan ukuran data sebesar 100 MB.

D-ITG nantinya digunakan pada sisi pengirim dan penerima. Pada sisi pengirim, D-ITG akan melakukan generate data paket sesuai dengan perintah yang diberikan dan mengirimnya ke alamat IP penerima. Perintah yang digunakan untuk mengirim traffic ditunjukkan pada gambar 3.12.

```
./ITGSend -T  TCP -a 8001::2 -c 34952 -C 30 -t 20000
-x receiver.log
```

Gambar 3.12 Konfigurasi D-ITG pada pengirim

Dimana, -a 8001::2 merupakan alamat ip dari komputer penerima, -TCP dan -x receiver1.log merupakan perintah untuk menyimpan hasil log dalam file receiver1.log pada komputer penerima. Sedangkan pada sisi penerima, software D-ITG melakukan listening packet data yang masuk. Perintah yang digunakan seperti pada gambar 3.13.

```
./ITGRecv
```

Gambar 3.13 Konfigurasi D-ITG pada penerima

3.4.1 Pengambilan Data pada Topologi Jaringan Konvensional

Data yang diambil pada jaringan SDN sama dengan pada jaringan kovensional yang meliputi *delay*, jitter, throughput dan packet loss. Pengambilan data dilakukan dengan aplikasi D-ITG yang terinstal pada *Host*. Pengambilan data dilakukan menggunakan beban *traffic* paket *UDP* dan *TCP* yang berukuran 20 MB, 40 MB, 60 MB, 80 MB, dan 100 MB. Pengiriman beban *traffic* dilakukan oleh *Host* 1 yang berperan sebagai pengirim dan diterima oleh *Host* 8 yang berperan sebagai penerima. Perhitungan

nilai *delay*, jitter, throughput dan packet loss dilakukan secara bersamaan. Setelah pengiriman beban *traffic* selesai nilai akan tersimpan pada file dengan format *.log yang dapat dibaca pada masing-masing *Host*.

3.4.2 Pengambilan Data pada Topologi Jaringan SDN

Data yang diambil pada jaringan *SDN* sama dengan pada jaringan kovensional yang meliputi *delay*, *jitter*, *throughput* dan *packet loss*. Pengambilan data dilakukan dengan aplikasi *D-ITG* yang terinstal pada *Host*. Pengambilan data dilakukan menggunakan beban *traffic* paket *UDP* dan *TCP* yang berukuran 20Mb, 40Mb, 60 Mb, 80Mb dan 100Mb. Pengiriman beban *traffic* dilakukan oleh *Host* 1 yang berperan sebagai pengirim dan diterima oleh *Host* 8 yang berperan sebagai penerima. Perhitungan nilai *delay*, *jitter*, *throughput* dan *packet loss* dilakukan secara bersamaan. Setelah pengiriman beban *traffic* selesai nilai akan tersimpan pada *file* dengan format *.log yang dapat dibaca pada masing-masing *Host*.