BAB 3
METODE PENELITIAN

Metodologi penelitian berisikan alur penelitian yang diwujudkan dalam bentuk *flowchart* penelitian, alat dan bahan, perancangan sistem. Penelitian ini menggunakan kuantitatif dan objektif. Metode yang digunakan menggunakan eksperimen untuk mengetahui penyebab dan akibat pada penelitian. Setelah melakukan eksperimen data dianalisis.

3.1 PERANGKAT YANG DIGUNAKAN

Dalam menunjang penelitian, kebutuhan menggunakan perangkat *hardware* maupun perangkat *software* pada penelitian yang akan dilakukan.

3.1.1 PERANGKAT KERAS (*HARDWARE*)

Perangkat keras yang dibutuhkan yaitu sebuah pc sebagai media untuk merealisasikan sistem yang akan dibangun. Spesifikasi perangkat keras yang digunakan ditunjukkan pada tabel 3.1.

Tabel 3.1 spesifikasi Perangkat

NO	Perangkat Keras	Spesifikasi
1	Sistem Operasi	Ubuntu 21.04
2	*Processor*	*Core i7*
3	RAM	8 GB
4	*Hardisk*	1 TB
5	*Type Graphics*	4 GB

3.1.2 PERANGKAT LUNAK (*SOFTWARE*)

Perangkat lunak diperlukan untuk membantu dalam merealisasikan sistem yang akan dibangun. Perangkat lunak yang digunakan sebagai tool dan aplikasi pada penelitian ditunjukkan pada tabel 3.2.

Tabel 3.2 spesifikasi *Software*

NO	Perangkat Lunak	Versi
1	Mininet	2.3.0
2	*Virtual Box*	6.1
3	RYU *Controller*	1.2
4	OpenDaylight *Controller*	16.0
5	*Open Flow*	1.5
6	*D-ITG*	2.8.1
7	*Wireshark*	2.6.10

3.2 *FLOWCHART* ALUR PENELITIAN

Diagram alur penelitian menjelaskan mengenai tahapan yang dilakukan pada penyusunan penelitian ini yang dijelaskan pada gambar diagram alur 3.1

Gambar 3.1 Diagram Alur Penelitian

Pada gambar 3.1 menunjukan diagram alur yang digunakan peneliti agar berjalan sesuai rencana dan dapat tercapai dengan baik. Penelitian dimulai dengan penentuan topologi jaringan sebagai alat untuk pengujian performansi *controller* Opendaylight dan RYU pada *software defined network* dengan *routing* OSPF. Setelah perancangan topologi adalah instalasi *software* yang akan digunakan untuk penelitian. *Software* yang digunakan meliputi *Virtualbox, Ubuntu 21.04,* Mininet, *Controller, D-ITG*. *Virtualbox* digunakan sebagai *virtual machine* yang didalamnya di *install* ubuntu. Penggunaan *virtualbox* digunakan untuk mempermudah peneliti dalam melakukan pengujian dan sebagai antisipasi ketika terdapat kegagalan konfigurasi. Setelah penginstalan selesai, Langkah selanjutnya adalah konfigurasi *controller* Opendaylight dan RYU dengan mininet dan dilakukan pengujian. Pengujian dilakukan dengan menjalankan topologi dan mengaktifkan *controller* Opendaylight dan RYU, diberikan perintah *pingall* untuk mengetahui topologi yang berisikan *host, switch* dan *controller* saling terhubung. Jika pengujian gagal maka dilakukan pengulangan konfigurasi *controller* dan dilakukan pengujian ulang sampai berhasil.

Tahapan selanjutnya adalah melakukan pengujian *controller* yang sudah dikonfigurasi dengan *routing* OSPF. Pengujian dilakukan dengan membuka 2 terminal pada salah satu *host* dan akan dijalankan D-ITG pengirim dan penerima. Pengujian yang dilakukan menghasilkan data hasil pengujian berupa QoS dan mendapatkan parameter *jitter, delay, throughput* dan *packet loss*. Pada pengujian konvergensi waktu dilakukan dengan memutuskan jalur yang tersambung pada *host* dan akan didapatkan nilai waktu dengan menggunakan bantuan *software wireshark* untuk mengetahui waktu yang dibutuhkan *controller* mendapatkan jalur baru Ketika terjadi pemutusan jalur. Hasil yang diperoleh selanjutnya dianalisis dan mendapatkan kesimpulan terhadap penelitian yang sudah dilakukan.

3.3 TOPOLOGI JARINGAN

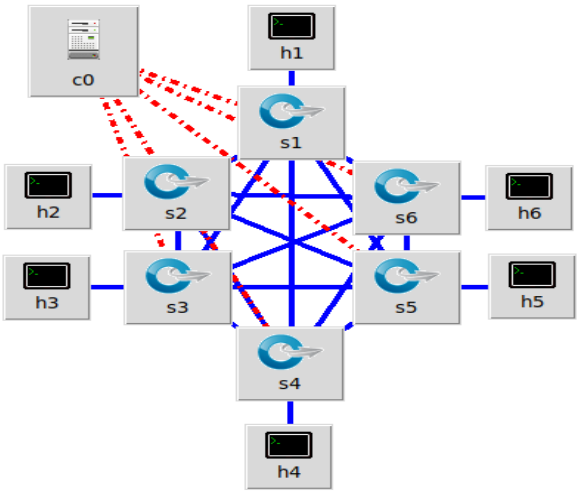
Topologi yang digunakan peneliti adalah topologi *mesh* seperti pada gambar 3.2 yang terdiri dari 6 buah *host*, 6 buah *switch* dan *controller*. *Controller* menggunakan *controller* Opendaylight dan RYU yang digunakan untuk mengatur komunikasi dan pertukaran data pada jaringan *software defined network*. Topologi *Mesh* diimplementasikan dengan menggunakan *emulator* mininet. Seperti pada

gambar 3.2 semua *host* saling terhubung dengan *host* lainya, karena seperti pada konsep topologi *full mesh* yang mana *host* saling terhubung ke semua *host*. Ketika jalur mengalami putus jalur, topologi *mesh* dapat membackup jaringan karena masih terdapat jalur lain yang dapat dilewati. Masing-masing *host* diberikan alamat IP seperti pada tabel 3.3 dari *host* satu sampai dengan *host* 6. Penetapan IP Address dimulai dari 10.0.0.1 sampai 10.0.0.6.

Tabel 3.3 *IP Address*

Host	***IP Address***
Host 1 (H1)	10.0.01
Host 2 (H2)	10.0.0.2
Host 3 (H3)	10.0.0.3
Host 4 (H4)	10.0.0.4
Host 5 (H5)	10.0.0.5
Host 6 (H6)	10.0.0.6

Topologi yang digunakan menggunakan topologi full mesh seperti pada gambar 3.2. terdapat 6 buah switch yang salih terhubung dengan switch lainya. Pada masing-masing *switch* memiliki 1 buah. Pada topologi penelitian menggunakan kabel berjenis ethernet.



Gambar 3.2 Topologi *Mesh* yang digunakan pada penelitian

3.4 SKENARIO PENGUJIAN

Skenario pengujian dibuat peneliti untuk mengetahui performansi dari penelitian yang dikerjakan. Pengujian diharapkan dapat memberikan hasil

performansi *routing* OSPF dan *controller* Opendaylight dan *RYU*. Skenario pengujian dilakukan dengan mengirimkan paket data dengan *protocol* TCP dan UDP. Scenario pengujian pada tabel 3.4 terdapat dua *protocol* yang diuji dengan besar data yang bervariasi, mulai dari 2 MB sampai dengan 10 MB. Waktu pengiriman diberikan sebesar 15 detik dan jumlah pengujian dilakukan sebanyak 10 kali. Pengujian dilakukan dengan menggunakan D-ITG untuk mendapatkan data dari parameter yang akan dianalisa. Sebelum menjalankan D-ITG *controller* dinyalakan terlebih dahulu dan dipastikan *routing* OSPF sudah berjalan. *Controller* menggunakan Opendaylight dan *RYU,* diujikan dengan bergantian. Mininet dijalankan untuk mengaktifkan topologi dan di sambungkan dengan *controller* yang sudah berjalan sebelumnya. Sebelum pengujian dilakukan *ping* antar *host* untuk memastikan masing-masing *host* sudah terhubung. Pada *host* 1 dan *host* 2 dilakukan *ping* sampai semua *host* dapat dipastikan terhubung. Pengujiang D-ITG dilakukan dengan membuka *xterm* pada terminal *host* pengirim penerima dan dilakukan pengujian sesuai skenario. D-ITG dijalankan pada 2 terminal host, satu host digunakan sebagai D-ITG pengirim dan *host* lain digunakan sebagai D-ITG penerima. Hasil D-ITG menghasilkan *log* yang berisikan *jitter*, delay, packet loss, *troughput*. Sedangkan untuk mengetahui nilai waktu konvergen dilakukan dengan memutus jalur atau link failure pada jalur s1-s2, s2-s3, s3-s4, s4-s5, s5-s6, s6-s1. pemutusan jalur dilakukan pada jalur utama atau jalur yangpaling dekat. Pemutusan jalur dilakukan untuk mengetahui seberapa cepat dari controller untuk menemukan jalur baru berdasarkan table routing yang sudah dibuat.

Tabel 3.4 Skenario Pengujian

No	Besar Data (MB)	Waktu Pengiriman (s)	Jumlah Pengujian
1	2	15	10
2	4	15	10
3	6	15	10
4	8	15	10
5	10	15	10

3.5 KONFIGURASI SIMULASI

3.5.1 Konfigurasi Mininet

Topologi jaringan mesh menggunakan perintah seperti pada gambar 3.3 dengan bentuk jaringan sesuai pada gambar 3.2. Pada perintah menjalankan

mininet membuka folder mininet dan menjalan *file miniedit.py*. *miniedit* merupakan *file* yang diberikan mininet untuk membuat atau menampilkan topologi secara GUI.

```
$ sudo mininet/examples/miniedit.py
```

Penulis membuat topologi sesuai pada gambar 3.2 dengan membuat 6 buah *host*, 6 buah *switch* dan 1 buah *controller*. Semua *host* dijadikan *client* untuk saling mengirimkan data dari masing-masing *host*. *Host* dan *switch* dihubungkan sesuai dengan topologi. *Switch* s1 dan s6 saling terhubung sehingga membentuk 16 jalur. Memberikan alamat IP pada *host* sesuai pada tabel 3.3 dari *host* 1 sampai 6. Pemberian alamat IP dengan mengedit *file* topologi yang sudah dibuat pada *miniedit*. Pada *host* 1 diberikan IP 10.0.0.1 sampai host yang terakhir seperti perintah berikut:

```
    info( '*** Add hosts\n')
    h6 = net.addHost('h6', cls=Host,
ip='10.0.0.6', defaultRoute=None)
    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)
    h4 = net.addHost('h4', cls=Host,
ip='10.0.0.4', defaultRoute=None)
    h3 = net.addHost('h3', cls=Host,
ip='10.0.0.3', defaultRoute=None)
    h2 = net.addHost('h2', cls=Host,
ip='10.0.0.2', defaultRoute=None)
```

3.5.2 Konfigurasi *Controller* RYU

Pada penelitian ini menggunakan *controller* RYU untuk uji performansi *routing* OSPF. Sebelum menjalankan *controller* dengan *routing* OSPF, dilakukan pengujian *controller* bahawasanya dapat berjalan tanpa *error*. Menjalankan *controller* RYU dengan memberikan perintah seperti perintah dibawah, perintah tersebut digunakan untuk mengaktifkan modul *controller* RYU.

```
# ryu-manager ~/ryu/ryu/app/simple_switch_stp_13.py
```

Berdasarkan perintah tersebut, setelah *controller* diaktifkan, selanjutnya menjalankan topologi *mesh* yang sudah dibuat dengan menggunakan perintah

dibawah. *File* topologi menggunakan bahasa *python* sehingga menjalankan *file* menggunakan perintah *python* 3 agar dapat dijalankan.

```
$ sudo mn --custom=mesh.py --topo=mytopo
```

Untuk pengujian *controller* dan topologi sudah terhubung, melakukan *ping* pada *host* 1 dan *host* 2. Modul *routing* OSPF dilakukan dengan membuat file routing pada controller folder *controller* RYU yang terletak pada ryu/lib/packet/OSPF_ryu.py. pada modul ospf bersikan konfigurasi routing ospf dalam berbahasa python. Seperti pada perintah

```
ospf = OSPFMessage
@OSPFMessage.register_type(OSPF_MSG_HELLO)
class OSPFHello(OSPFMessage):

   _PACK_STR = '!4sHBBI4s4s'  # + neighbors
   _PACK_LEN = struct.calcsize(_PACK_STR)
   _MIN_LEN = OSPFMessage._HDR_LEN + _PACK_LEN

   def __init__(self, length=None, router_id', area_id,
                au_type=1,          authentication=0,          checksum=None,
version=_VERSION,
                mask, hello_interval=10, options=0, priority=1,
                dead_interval=40, designated_router,
                backup_router, neighbors=None):
       neighbors = neighbors if neighbors else []
       super(OSPFHello,      self).__init__(OSPF_MSG_HELLO,      length,
router_id,
                              area_id, au_type, authentication,
                                                          checksum,
version)
```

Pada penggalan isi modul ospf tersbut terdapat kelas yang Bernama OSPF *hello*. Kelas tersebut digunakan untuk menentukan router tetangga yang ada pada suatu jaringan. Pesan hello berisikan dengan router id dan juga area id tetangga. Pengriman paket hello dilakukan dengan interval 10 detik untuk menjalankan Modul OSPF pada *controller* RYU diilustrasikan pada perintah berikut:

```
$ ryu-manager OSPF_ryu.py --observer-links
```

Pengujian modul *routing* OSPF pada RYU *controller* dilakukan dengan *ping host* ke *host* lainnya. Seperti pada gambar 3.3 diujikan *ping* dari h1 ke h2 dengan alamat IP 10.0.0.2. Paket icmp yang dikirimkan dari h1 menuju h2 berhasil tanpa terjadi *Request Time Out.*

```
mininet> h1 ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.862 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.131 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.129 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.127 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.126 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.144 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.128 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.125 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.122 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.129 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.124 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.126 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=0.128 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=0.127 ms
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=0.125 ms
64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=0.119 ms
64 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=0.162 ms
```

3.5.3 Konfigurasi *Controller* Opendaylight

Pada penelitian ini menggunakan *controller* Opendaylight untuk uji performansi *routing* OSPF. Sebelum menjalankan *controller* dengan *routing* OSPF, dilakukan pengujian *controller* bahawasanya dapat berjalan tanpa *error*. Perintah -*E karaf* merupakan perintah untuk mengaktifkan *controller* opendaylight dan mengaktifkan *odl-l2switch-switch-ui*. *Controller* opendaylight memiliki tampilan *web* UI dengan membuka di *browser* menggunakan alamat *localhost* atau menggunakan alamat *loopback*. Pengujian dilakukan dengan menjalankan *controller* opendaylight dan mengaktifkan topologi pada mininet. *sudo mn* digunakan untuk membuka mininet dan menjalankan topologi dengan format *python*. *Controller* dihubungkan dengan menambahkan IP 127.0.0.1 dengan *port* 6633. Perintah untuk menjalankan topologi seperti yang ditunjukkan pada perintah berikut:

```
$ sudo mn --custom=mesh.py --topo=mytopo --
controller=remote,ip=127.0.0.1,port=6633
```

Pertintah tersebut membuka topologi mesh dan menymabungkan controller opendaylight yang sudah berjalan dengan menggunakan ip loopback 127.0.0.1 dengan port 6633. Konfigurasi OSPF dengan membuat modul routing ospf pada folder controller Opendaylight. Modul berisikan routing OSPF denganmenggunakan Bahasa java. Modul tersebut akan memberikan routing ospf

dari memberikan paket hello untuk mengetahui router sekitar, megirimkan paket DBD, LSR, LSU dan LSack. *File* modul ospf berisikan

```
private void Process_Hello(OSPF_Hello hello, OSPF_Interface oif, int
src)
{
int twoway = 0;
int i;
if (isDebugEnabled() && isDebugEnabledAt(DEBUG_NEIGHBOR))
debug("----- LS_HELLO_RECEIVED at if " + oif.if_id + " from "
            + src + ": " + hello);
   /* HelloInterval check */
   int hello_interval = hello.get_hello_interval();
   if (hello_interval!= oif.hello_interval) {
     if (isErrorNoticeEnabled())
       error("Process_Hello()", "*** Warn *** : "
              + "HelloInterval mismatch");
     return;
   }
  /* RouterDeadInterval check */
   int router_dead_interval = hello.get_router_dead_interval();
    if (router_dead_interval!= oif.dead_interval) {
      if (isErrorNoticeEnabled())
        error("Process_Hello()", "*** Warn *** : "
               + "RouterDeadInterval mismatch with ");
      return;
    }
```

Pada penggalan modul tersebut merupakan perintah untuk saling mengirimkan paket hello untuk menegetahui perangkat yang ada pada jaringan. Paket hello akan dikirmmkan dengan interval 10. Paket hello dikatakan berhasil Ketika paket tersebut berhasil mengirimakn id router dan Ketika tidak menerimanya aka nada dalam keadaan mismatch. Pengiriman ulang dilakukan sampai mengetahui semua perangkat tetangganya. Untuk menjalankan modul tersebut dengan mengetikan perintah.

```
$ sudo E - Karaf
```

Perintah tersebut untuk mengaktifkan *controller* Opendaylight sekaligus mengaktifkan modul *routing* OSPF yang sudah dibuat sebelumnya. Pengujian dilakukan dengan *ping host* ke *host* lainnya. Seperti pada perintah dibawah diujikan *ping* dari h1 ke h2 dengan alamat IP 10.0.0.2. Paket icmp yang dikirimkan dari h1 menuju h2 berhasil tanpa terjadi *Request Time Out.*

```
mininet> h1 ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.862 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.131 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.129 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.127 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.126 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.144 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.128 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.125 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.122 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.129 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.124 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.126 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=0.128 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=0.127 ms
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=0.125 ms
64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=0.119 ms
64 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=0.162 ms
```