BAB 4 PERANCANGAN DAN IMPLEMENTASI

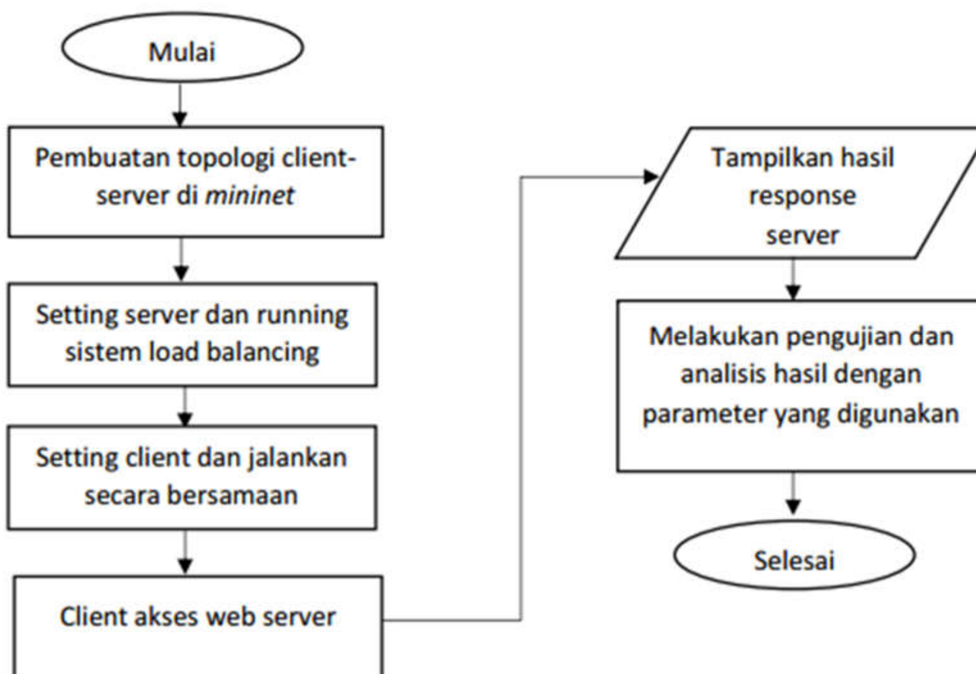
Pada bagian ini akan dijelaskan tentang langkah-langkah perancangan dan mengimplementasikan *algoritme shortest delay* dengan load balancing pada *openFlow software defined network*.

4.1 Perancangan

Perancangan yang dibutuhkan sebagai tahap perencanaan terhadap sistem yang akan dibangun sesuai dengan bab-bab sebelumnya mengenai persyaratan kebutuhan sistem.

4.1.1 Diagram Alir Perancangan Sistem

Pada bagian ini akan dijelaskan perancangan sistem dimulai dengan tahapan pembuatan topologi sampai hasil pengujian hingga kesimpulan yang dilakukan terhadap sistem.



Gambar 4. 1 Flowchart Perancangan Sistem

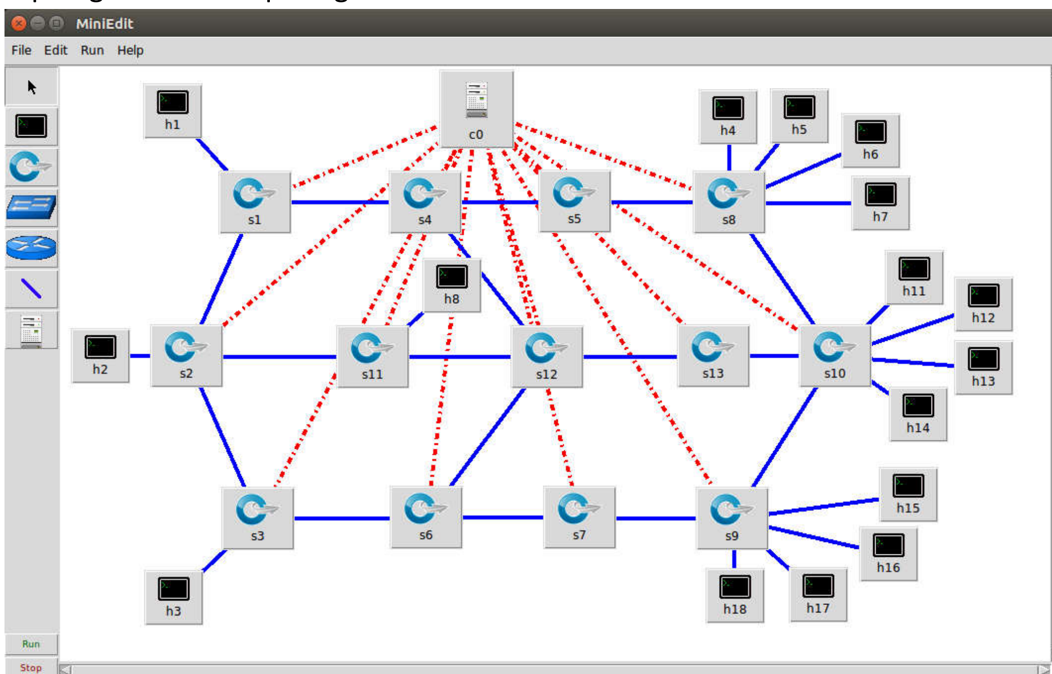
Pada gambar 4.1 di atas dapat diketahui bahwa untuk tahapan rancangan implementasi sistem dimulai dengan pembuatan topologi untuk *client-server* pada mininet yang nantinya digunakan untuk melakukan implementasi *load-balancing*. Pada pengujian ini terdapat 16 buah host yang masing-masing terdiri dari 3 host sebagai server dan 13 host lainnya sebagai client. Pengujian ini juga dilakukan dengan memberikan beban ke server seperti *disturbance conditions* di sisi client dimana untuk mengetahui pendistribusian beban ke beberapa server ketika

banyak permintaan dari client. Kemudian melakukan *setting server*. Selanjutnya merupakan proses komunikasi yang dilakukan oleh *client* dengan mengakses *web server*. Setelah proses permintaan *user* terhadap *web server* selesai dilakukan maka akan ditampilkan hasil dari sistem *load balancing* menggunakan algoritme *shortest delay*. Pengujian ini juga dilakukan dengan membandingkana algoritme *shortest delay* dan algoritme *round-robin*. Tahapan akhir yaitu melakukan pengujian sistem dan melakukan analisis hasil pengujian dengan berbagai parameter.

Perancangan sistem untuk pembuatan topologi akan didasarkan pada bab sebelumnya yang telah dibahas. Berdasarkan penjelasan jumlah *host* yang digunakan, jumlah *host client* dan jumlah *host server.* Untuk mekanisme pemilihan *server* berdasarkan algoritme *shortest delay* akan dilakukan monitoring melalui *controller* yang dimiliki oleh konsep dasar *software defined network*. Selanjutnya akan dilakukan pengujian sistem *load balancing* dengan berbagai parameter dan dilakukan analisis hasil nya. Beberapa parameter yang digunakan dalam pengujian sistem meliputi *throughput, connection rate* dan *reply time*

4.1.2 Topologi

Tahap awal dimulai dengan membangun topologi jaringan *client server* pada mininet yang nantinya digunakan untuk melakukan implementasi *load balancing* dengan mengalokasikan ke server dengan delay terendah. Maka, diperlukan suatu topologi yang dapat mengalokasikan permintaan user ke server dengan delay terendah sehingga memberi respon yang cepat. *Topologi mesh* merupakan bentuk topologi yang dimana setiap perangkat terhubung antara perangkat yang satu dengan perangkat lainnya didalam jaringan. Rancangan topologi tersebut seperti gambar berikut ini:

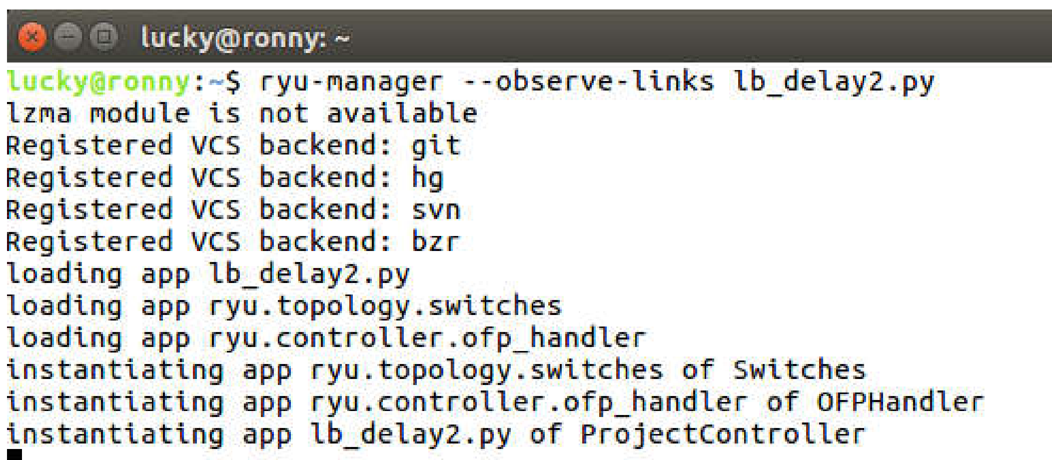


Gambar 4. 2 Topologi Penelitian di *Mininet*

Dari gambar 4.2 dapat di jelaskan *create topologi* sebanyak 16 host. Kemudian menambahkan *controller* (c0) dan *single switch* sebanyak 13 switch. Untuk menghubungkan antar host dan switch dilakukan *create link* dari (h1,s1), (h2,s2), (h3,s3), (h8,s11), (h4,h5,h6,h7,s8), (h11,h12,h13,14,s10) dan (h15,h16,h17,h18,s9). Untuk pengujian ini dilakukan setting pada link bandwitch sebesar 100 Mbit antar switch. Setelah semua topologi selesai dibuat *controller* dan *switch* akan melakukan *starting* sistem.

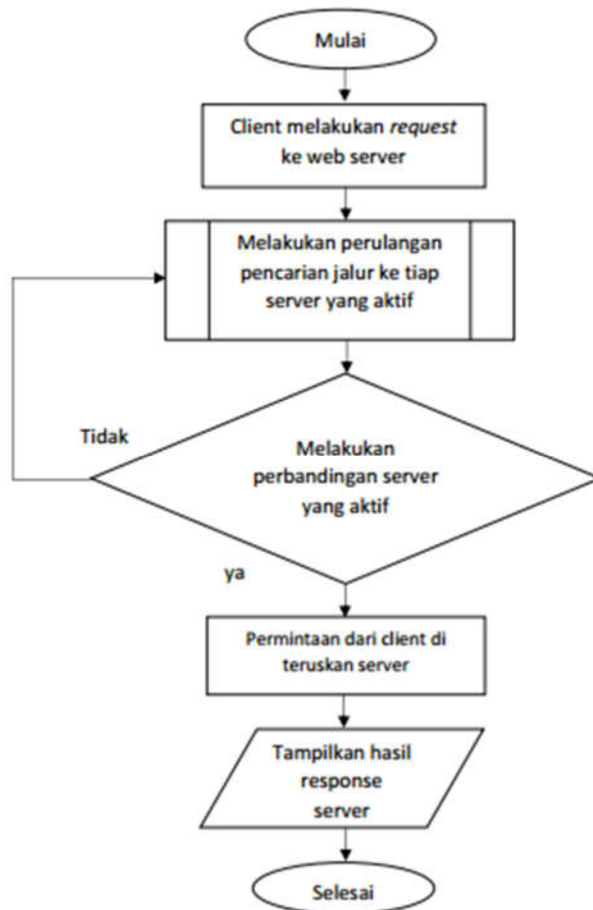
4.1.3 Load Balancing

Dalam melakukan *implementasi load balancing pada web server menggunakan algoritme shortest delay pada software defined network* dilakukan running sistem load balancing menggunakan *ryu controller.* Perintahnya sebagai berikut:



Gambar 4. 3 Perintah jalankan *Ryu-controller*

Pada gambar 4.3 dapat diketahui perintah tersebut digunakan untuk menjalankan load balancing dengan *ryu-controller* dengan nama program *"lb_delay.py"*. Perintah di atas untuk menjalankan algoritme *shortest delay.* Pada pengujian ini juga akan dibandingkan dengan algoritme *round-robin* dengan nama program *"lb_roundrobin_ok.py.* Sehingga program controller load balancing siap untuk di jalankan. Pada perancangan sistem ini juga menggunakan metode pemilihan server dengan load balancing. Berikut blok-diagram yang menjelaskan tentang load balancing yang akan diterapkan:



Gambar 4. 4 *Flowchart* Load Balancing

Berdasarkan Gambar 4.4 di atas di ketahui proses *load balancing*, dapat diketahui bahwa *load balancing* merupakan mekanisme pembagian beban untuk *server* yang akan membagi *traffic request* yang dilakukan oleh *client*.

Proses yang terjadi sesuai *flowchart* di atas diawali dengan proses untuk mengeksekusi jumlah *host* yang akan digunakan. Client melakukan *request* ke web server selanjutnya *request* tersebut akan dilakukan pengecekan untuk pemilihan jalur ke tiap server. Pengecekan dilakukan dengan cara melakukan perbandingan untuk mencari jalur menuju ke server yang memiliki *delay* terendah. Setelah melakukan pengecekan, permintaan client akan di teruskan ke server yang memliki *delay* terendah sehingga server akan memberikan respon yang cepat. Kemudian tampilkan hasil response server.

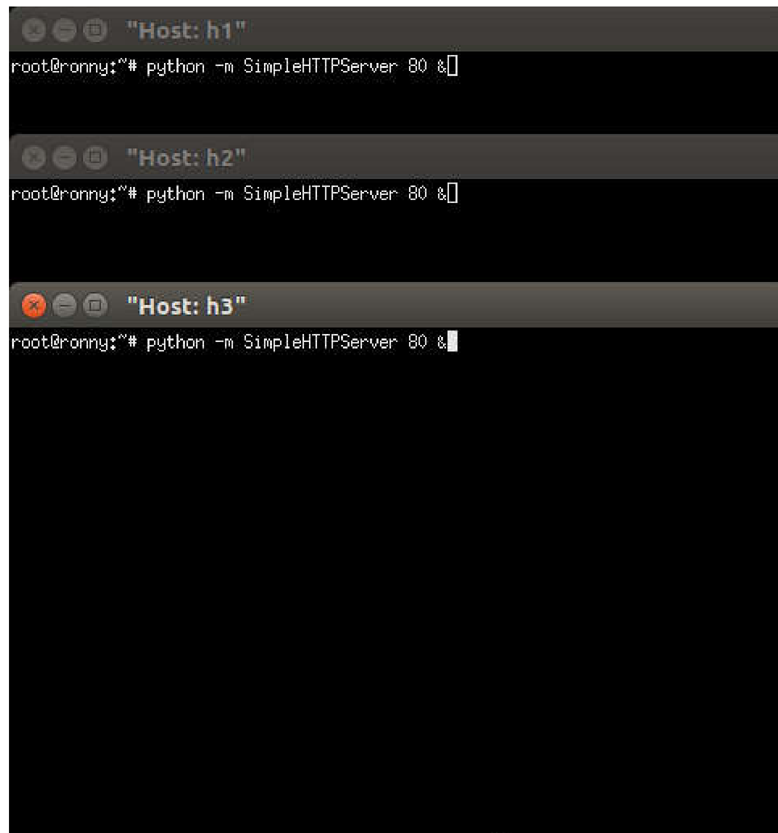
4.1.3.1 Setting Server

Dalam pengujian ini server yang digunakan sebanyak 3 buah yaitu h1, h2 dan h3. Untuk melakukan pengaturan server pada host digunakan perintah sebagai berikut:

```
root@ronny:~# python -m SimpleHTTPServer 80 &
```

Gambar 4. 5 Commend-line Setting Server

Pada gambar 4.5 menunjukkan perintah di atas digunakan untuk melakukan *setting* server pada host dengan perintah protocol HTTP port 80. Berikut hasil dari perintah tersebut:



Gambar 4. 6 *Setting* Server

Pada gambar 4.6 dapat di ketahui pengaturan server pada host sudah selesai dan server siap digunakan pada protocol HTTP port 80.

4.1.3.2 Setting Client

Dalam melakukan pengujian load balancing pada web server diperlukan *client* untuk *request*. *Client* yang digunakan sebanyak 13 host di antaranya 1 host, yaitu h8 digunakan untuk beban traffic ke server sebab dengan adanya *disturbance conditions* dapat mengetahui performa web server dalam menangani banyak *request* dari client sedangkan 12 host, yaitu h4 sampai h7 dan h11 sampai h18 nantinya berfungsi sebagai client untuk melakukan permintaan ke web server.

Gambar 4. 7 *Commend-line* Setting Client

Pada gambar 4.7 di atas adalah tampilan pada *host 4* yang digunakan untuk melakukan permintaan ke web server. Perintah di atas untuk menguji web server menggunakan *httperf* di sisi client. Beberapa penjelasan tentang source diatas:

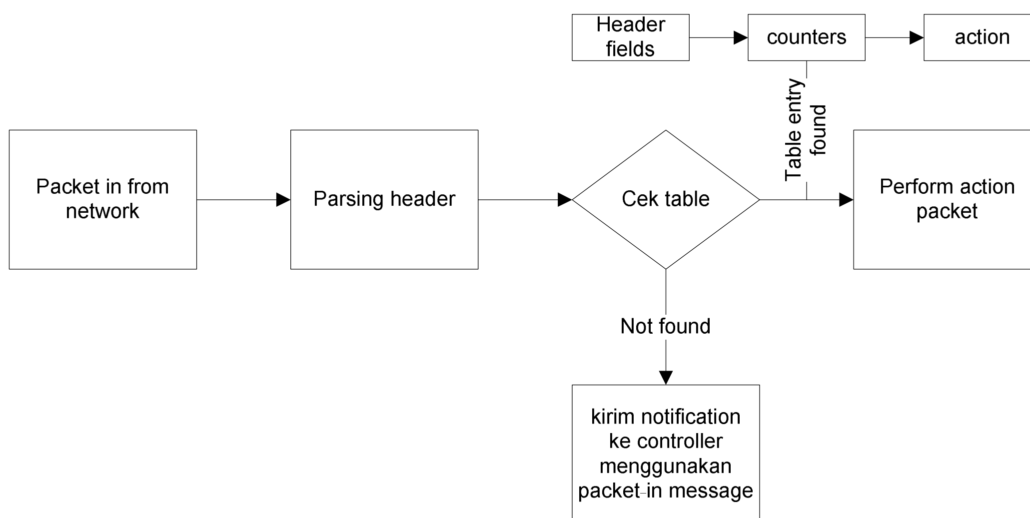
Tabel 4. 1 Penjelasan Commend-line *Client Httperf*

Parameter	Keterangan
Httperf	Perintah menjalankan httperf
--uri=/nama_picture.jpg	Untuk mengakses file picture dengan format jpg
--num-conn=60	Menginstruksikan httperf untuk menyediakan 60 koneksi
-*request/second*=6	Membuat 6 koneksi baru per detik

Pada tabel 4.1 menunjukkan *commend-line* httperf dan digunakan untuk menguji kinerja *httperf.*

4.1.3.3 *Flowchart* Pengecekan Packet

Berikut merupakan pengecekan *flowchart* pengecekan paket *forwarding* dengan *openflow* di switch.



Gambar 4. 8 *Blok-Diagram* Pengecekan Paket

Sumber : (McKeown, 2008)

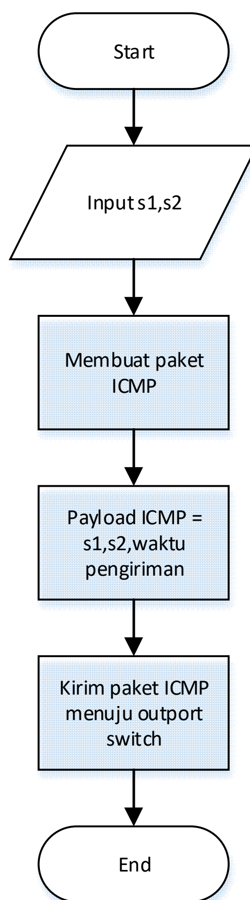
Berdasarkan Gambar 4.8 di atas maka dapat disimpulkan bahwa ketika ada paket dari suatu jaringan maka akan diproses *parsing header* yang dimiliki paket tersebut. *Header* paket berisi tentang informasi paket. Selanjutnya dilakukan pengecekan pada *tabel openflow* untuk dilakukan *action* terhadap paket tersebut. Jika dalam *tabel openflow* paket sesuai maka *switch* akan meneruskan paket menuju *destination*, sedangkan jika tidak ada kesesuaian *header* paket maka *switch* akan mengirimkan pemberitahuan kepada *controller* untuk memberikan *action* terhadap paket tersebut. Setiap paket yang datang dalam suatu jaringan

maka diberikan action terhadap paket tersebut sesuai dengan *flow tabel* yang diatur oleh *controller*.

Mekanisme pengecekan tabel berdasarkan tiga komponen yaitu *header fields, counters dan action*. *Header fields* merupakan pengecekan *header* yang dimiliki oleh paket. *Counter* merupakan jumlah paket dan *byte* yang dimiliki oleh setiap *flow.* Sedangkan *action* merupakan aktifitas yang diberikan terhadap paket. Misalnya paket akan diteruskan menuju ke tujuan ataupun paket akan dibuang karena tidak sesuai dengan tabel *flow* paket.

4.1.4 Perancangan Monitoring Delay

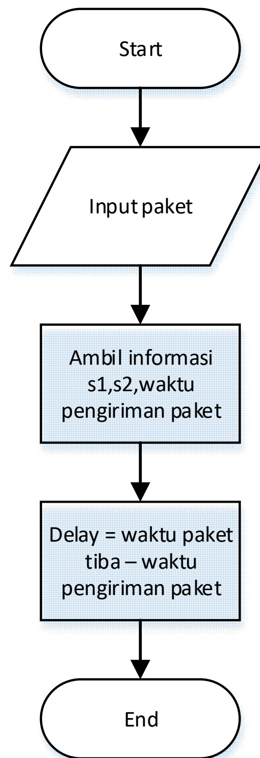
Pada bagian ini berisi tentang bagaimana membangun sebuah fungsi yang digunakan untuk memantau *delay* pada setiap *link.* Nilai *delay* didapatkan dengan cara mengirim paket ICMP menggunakan packet library pada Ryu. Diagram alir perancangan monitoring *delay* seperti pada gambar berikut ini.



Gambar 4. 9 Membuat Paket ICMP

Untuk dapat mengirimkan paket ICMP, diperlukan informasi mengenai switch yang bertetangga (“s1” dan “s2”). Kemudian controller akan membuat paket ICMP menggunakan packet library yang ada pada Ryu. *Payload* pada paket tersebut akan diisi data-data yang dibutuhkan dalam pencarian delay seperti informasi pasangan switch yang bertetangga dan waktu pengiriman paket

tersebut. Kemudian paket tersebut akan dikirimkan menuju outport pada switch tersebut. Disisi lain terdapat fungsi yang digunakan untuk meng-handle paket ICMP ketika paket tersebut sampai pada switch tujuan. Diagram alir fungsi ICMP *handler* seperti pada gambar di bawah ini.



Gambar 4. 10 Paket ICMP *handler*

Ketika paket ICMP masuk pada switch tujuan, maka *payload* paket tersebut akan diekstrak sehingga menghasilkan data-data yang diperlukan dalam pencarian delay seperti informasi switch yang bertetangga dan waktu pengiriman paket. Berdasarkan data tersebut, *delay* dari switch yang bertetangga ("s1" dan "s2") didapatkan dengan cara menghitung selisih antara waktu saat paket tiba pada switch tujuan dengan waktu saat pengiriman dari switch asal.

4.1.5 Web Server

Untuk melakukan implementasi load balancing pada web server digunakan perintah *#curl* pada alamat *IP 10.0.0.20* yang dilakukan pada sisi client. Pada pengujian ini menggunakan *ip virtual* dengan alamat *IP 10.0.20*. *#curl* berfungsi untuk mengecek apakah alamat ip tersebut berjalan atau tidak sebagai ip web server. Kemudian akan muncul *doctype html* yang nantinya data tersebut digunakan untuk melakukan permintaan ke server.

```
"Host: h4"
root@ronny:~# curl 10.0.0.20
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><html>
<title>Directory listing for /</title>
<body>
<h2>Directory listing for /</h2>
<hr>
<ul>
<li><a href=".bash_history">.bash_history</a>
<li><a href=".bash_logout">.bash_logout</a>
<li><a href=".bashrc">.bashrc</a>
<li><a href=".cache/">.cache/</a>
<li><a href=".compiz/">.compiz/</a>
<li><a href=".config/">.config/</a>
<li><a href=".dbus/">.dbus/</a>
<li><a href=".dmrc">.dmrc</a>
<li><a href=".gconf/">.gconf/</a>
<li><a href=".gnupg/">.gnupg/</a>
<li><a href=".ICEauthority">.ICEauthority</a>
<li><a href=".local/">.local/</a>
<li><a href=".mininet_history">.mininet_history</a>
<li><a href=".mozilla/">.mozilla/</a>
<li><a href=".nano/">.nano/</a>
<li><a href=".PlayOnLinux/">.PlayOnLinux/</a>
<li><a href=".profile">.profile</a>
<li><a href=".rnd">.rnd</a>
<li><a href=".ssh/">.ssh/</a>
<li><a href=".sudo_as_admin_successful">.sudo_as_admin_successful</a>
<li><a href=".viminfo">.viminfo</a>
<li><a href=".wget-hsts">.wget-hsts</a>
<li><a href=".wireshark/">.wireshark/</a>
<li><a href=".Xauthority">.Xauthority</a>
<li><a href=".xsession-errors">.xsession-errors</a>
<li><a href=".xsession-errors.old">.xsession-errors.old</a>
<li><a href="Bismillah/">Bismillah/</a>
<li><a href="build/">build/</a>
<li><a href="Desktop/">Desktop/</a>
<li><a href="Documents/">Documents/</a>
```

Gambar 4. 11 Web server

Pada gambar 4.11 di atas dapat di ketahui bahwa permintaan *client* dengan menggunakan *#curl* telah dikirim oleh server berupa halaman *doctype html*.

4.2 Implementasi

Tahapan yang dikerjakan pada subbab implementasi mengikuti metodologi penelitian yang telah dibahas, yaitu seperti berikut:

4.2.1 Instalasi

Instalasi memuat langkah-langkah yang dilakukan untuk memasang perangkat lunak pendukung untuk melakukan pengembangan sistem. Berikut ini beberapa perangkat lunak pendukung dengan cara instalasinya, yaitu sebagai berikut:

4.2.1.1 Mininet

Mininet adalah emulasi jaringan yang digunakan untuk pengembangan pada sistem di penelitian ini. Berikut ini dijelaskan bagaiman cara melakukan instalasi Mininet pada *Operating sistem Ubuntu*.

1. Meng-unduh *source code Mininet* pada *github Mininet* dengan perintah berikut pada terminal:

$ git clone git://github.com/mininet/mininet

2. Untuk memulai proses installasi *Mininet* dengan perintah:

   $ sudo mininet/util/install.sh –a

4.2.1.2 Ryu Controller

Untuk melakukan instalasi *Ryu* sebagai *Controller* pada jaringan *OpenFlow*, Berikut ini langkah-langkah yaitu dapat dilakukan:

1. Melakukan instalasi *python-pip* terlebih dahulu dengan menjalankan perintah berikut pada terminal:

   $ sudo apt-get install python-pip

2. Untuk memulai proses instalasi *ryu* dengan perintah:

   $ sudo pip install ryu

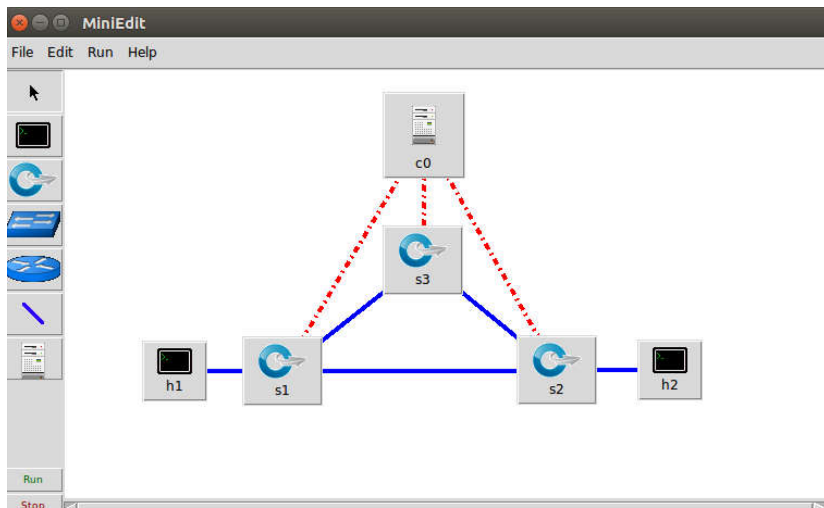
4.2.2 Membangun Topologi di Mininet

Setelah semua perangkat lunak pendukung telah terinstall, langkah selanjutnya adalah membangun topologi di *Mininet* sesuai dengan desain topologi yang telah dilakukan. Salah satu cara untuk melakukan pembangunan topologi di *Mininet* adalah dengan menggunakan GUI yang bernama *Miniedit*, yang akan dibahas di bagian ini. Berikut ini langkah-langkah membangun topologi di *Miniedit*, yaitu sebagai berikut :

1. Untuk Menjalankan *Miniedit* jalankah perintah berikut pada terminal:

   $ sudo python mininet/examples/miniedit.py

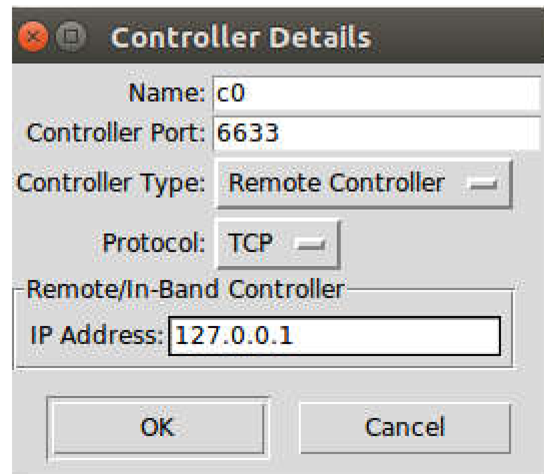
2. Membangun topologi. Salah satu contoh hasil pembangunan topologi dapat dilihat pada gambar 4.12.



Gambar 4. 12 Contoh Membangun Topologi di *Miniedit*

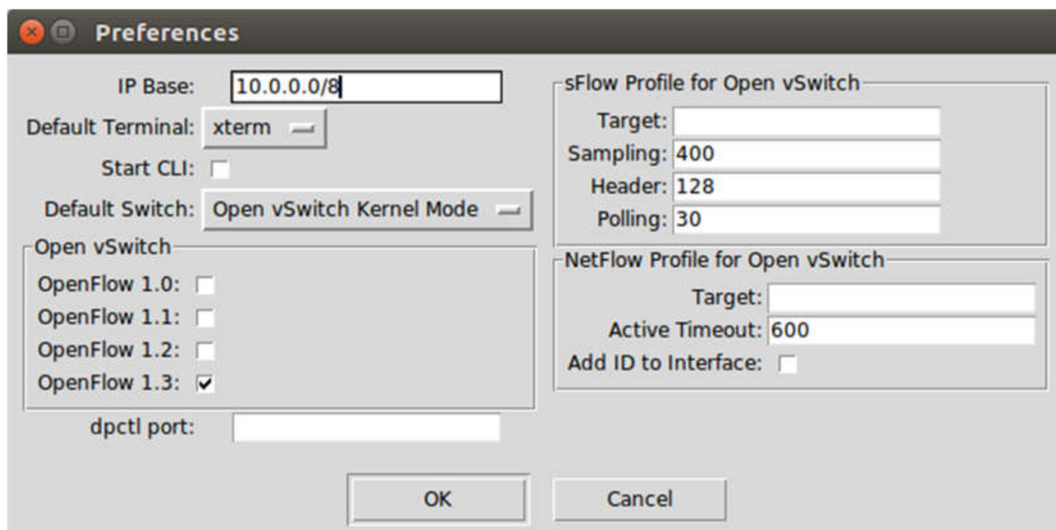
3. Melakukan pengaturan *controller* seperti gambar 4.13 dengan cara klik kanan pada logo *controller* (c0) kemudian memilih *properties*. Hal ini

diperlukan agar nantinya *Mininet* dapat terhubung dengan *controller*. Pada *"Controller Type"* ubah menjadi "*Remote Controller*".



Gambar 4. 13 Pengaturan Controller di *Miniedit*

4. Melakukan pengaturan pada *preferences* di *Miniedit* untuk mengaktifkan mengubah versi *OpenFlow* menjadi versi 1.3 seperti pada gambar 4.14. Hal ini dilakukan dengan cara memilih menu Edit kemudian memilih *Preferences*.



Gambar 4. 14 Pengaturan *Prefrences* di Mininet

5. Untuk menjalankan simulasi dengan cara menekan tombol "*Run*".
6. Membuka terminal yang baru untuk menjalankan program *controller* dengan melakukan perintah berikut:

   $ sudo ryu-manager --observe-links [nama_program].py

4.2.3 Pengembangan Program Controller

Pengembangan program controller terdiri dari langkah-langkah pemrograman yang dilakukan dalam Ryu Controller dengan menggunakan Bahasa python versi 2.7.11 dalam mengimplementasikan logika atau kecerdasan dari berdasarkan perancangan yang telah dilakukan.

4.2.3.1 Sourcode Load Balancing Shortest Delay

Dalam melakukan *implementasi load balancing pada web server menggunakan algoritme shortest delay pada software defined network* dalam pemilihan *server* yang dilakukan oleh controller. Sistem load balancing *algoritme shortest delay source code* sebagai berikut:

```
Algoritme 1 : Code Shortest Delay
1.  for server in self.servers:
2.  ip_server = server
3.  mac_server = self.arp_table[ip_server]
4.  p,d = get_path(mymac[eth.src][0], mymac[mac_server][0],
5.                 mymac[eth.src][1], mymac[mac_server][1])
6.  print p, d
7.  if d < minimum:
8.         minimum = d
9.         path = p
10.        selected_server_ip = server
11. print "Selected server %s" % selected_server_ip #
12. print path, minimum
```

Pada code di atas menjelaskan bagaimana mekanisme *shortest delay* akan berjalan, yaitu sebagai berikut:

- Pada baris ke-4 sampai baris ke-5 digunakan untuk mencari jalur menuju tiap server. Parameter yang dikirim adalah mac address dan switch_id.
- Pada baris ke-6 variabel (p) merupakan path dan varibel (d) merupakan bobot keseluruhan jalur.
- Pada baris ke-7 sampai baris ke-10 digunakan untuk mencari bobot delay paling kecil dari jalur semua server.

4.2.3.2 Sourcode Monitoring Delay

Untuk dapat menghitung parameter *delay*, dibutuhkan informasi tentang *delay* yang ada pada suatu *link*. Informasi tersebut dapat diambil dengan memanfaatkan Library Ryu yaitu dengan cara mengirimkan paket ICMP ke semua switch yang bertetangga dengan dirinya. Berikut ini merupakan implementasi dari fungsi *monitoring delay* dengan Library Ryu di tunjukkan sebagai berikut:

Code monitoring delay	
1.	def monitor_link(self, s1, s2):
2.	while True:
3.	self.send_ping_packet(s1, s2)
4.	time.sleep(0.5)
5.	self.logger.info('Stop monitoring link %s %s' %
6.	(s1.dpid, s2.dpid))
7.	
8.	def send_ping_packet(self, s1, s2):
9.	datapath = self.datapath_list[int(s1.dpid)]
10.	dst_mac = self.ping_mac
11.	dst_ip = self.ping_ip
12.	out_port = s1.port_no
13.	actions =[datapath.ofproto_parser.OFPActionOutput(out_port)]
14.	pkt = packet.Packet()
15.	
16.	pkt.add_protocol(ethernet.ethernet(ethertype=ether_types.ETH_
17.	TYPE_IP,
18.	src=self.controller_mac,
19.	dst=dst_mac))
20.	pkt.add_protocol(ipv4.ipv4(proto=in_proto.IPPROTO_ICMP,
21.	src=self.controller_ip,
22.	dst=dst_ip))
23.	echo_payload = '%s;%s;%f' % (s1.dpid, s2.dpid, time.time())
24.	payload = icmp.echo(data=echo_payload)
25.	pkt.add_protocol(icmp.icmp(data=payload))
26.	pkt.serialize()
27.	
28.	out = datapath.ofproto_parser.OFPPacketOut(
29.	datapath=datapath,
30.	buffer_id=datapath.ofproto.OFP_NO_BUFFER,
31.	data=pkt.data,
32.	in_port=datapath.ofproto.OFPP_CONTROLLER,
33.	actions=actions
34.	)
35.	
36.	datapath.send_msg(out)
37.	
38.	def ping_packet_handler(self, pkt):
39.	icmp_packet = pkt.get_protocol(icmp.icmp)
40.	echo_payload = icmp_packet.data

41.	payload = echo_payload.data
42.	info = payload.split(';')
43.	s1 = info[0]
44.	s2 = info[1]
45.	latency = (time.time() - float(info[2])) * 1000 # in ms
46.	# print "s%s to s%s latency = %f ms" % (s1, s2, latency)
47.	delay[int(s1)][int(s2)] = latency
48.	delay[int(s2)][int(s1)] = latency

Pada code di atas menjelaskan bagaimana *monitoring delay* akan berjalan, yaitu sebagai berikut:

- Baris ke-1 sampai baris ke-4 digunakan untuk memonitoring delay setiap 0.5 detik dengan cara memanggil fungsi send_ping_paket dengan parameternya yaitu switch yang bertetangga ("s1" dan "s2").
- Baris ke-8 merupakan fungsi yang digunakan untk mengirim paket dengan parameter ("s1" dan "s2").
- Baris ke-23 terdapat payload paket ICMP yang berisi informasi switch asal,switch tujuan, dan waktu pengiriman.
- Baris ke-38 terdapat fungsi yang digunakan untuk meng-handle saat paket ICMP tiba di switch tujuan.
- Baris ke-39 sampai baris ke-44 digunakan untuk mengambil data yang akan digunakan dalam perhitungan delay, seperti switch asal(s1), switch tujuan(s2) dan waktu pengiriman paket.
- Baris ke-45 berisi perhitungan delay dengan cara mengurangi waktu saat paket tiba dengan waktu pengiriman paket. Kemudian akan dikalika dengan 1000 untuk mengubah dari detik ke mili detik.
- Baris ke-47 sampai baris ke-48 mengisi variabel global "delay" sesuai dengan pasangan switch nya.

4.2.3.3 Sourcode Algoritme Pencarian jalur

Dalam melakukan *implementasi load balancing pada web server menggunakan algoritme shortest delay pada software defined network* menggunakan pencarian rute untuk menentukan jalur terpendek antar switch.

Code Pencarian Jalur	
1.	def minimum_cost(cost, Q):
2.	min = float('Inf')
3.	node = 0
4.	for v in Q:
5.	if cost[v] < min:
6.	min = cost[v]
7.	node = v

```
8.          return node
9.
10.     def get_path(src, dst, first_port, final_port):
11.         cost = defaultdict(lambda: float('Inf'))
12.         previous = defaultdict(lambda: None)
13.
14.         cost[src] = 0
15.         Q = set(switches)
16.
17.         while len(Q) > 0:
18.             u = minimum_cost(cost, Q)
19.             Q.remove(u)
20.             for p in switches:
21.                 if adjacency[u][p] != None:
22.                     w = delay[u][p]
23.                     if cost[u] + w < cost[p]:
24.                         cost[p] = cost[u] + w
25.                         previous[p] = u
26.                         return previous
```

Berdasarkan code diatas dapat dijelaskan jalan program *pencarian jalur* sebagai berikut:

- Baris ke-1 sampai baris ke-8 terdapat fungsi *"minimum_distance"* yang digunakan untuk mecari jarak switch paling kecil.
- Baris ke-11 digunakan untuk menginisialisasi variabel jarak switch ("cost") menjadi tak terhingga.
- Baris ke-12 digunakan untuk menginisialisasi variabel predecessor ("previous") menjadi 'None'.
- Baris ke-14 digunakan untuk mengatur jarak switch asal menjadi 0.
- Baris ke-15 merupakan variabel (Q) yang berisi seluruh switch pada topologi.
- Baris ke-17 melakukan perulangan sampai seluruh switch dikunjungi.
- Baris ke-18 memanggil fungsi *"minimum_distance"*.
- Baris ke 19 mengurangi switch yang telah dikunjungi.
- Baris ke-20 sampai baris ke-25 digunakan untuk update jarak dengan bobot yang diketahui.
- Baris ke26 digunakan untuk mengembalikan nilai berupa predecessor ("previous") tiap switch.