BAB 4 PERANCANGAN DAN IMPLEMENTASI

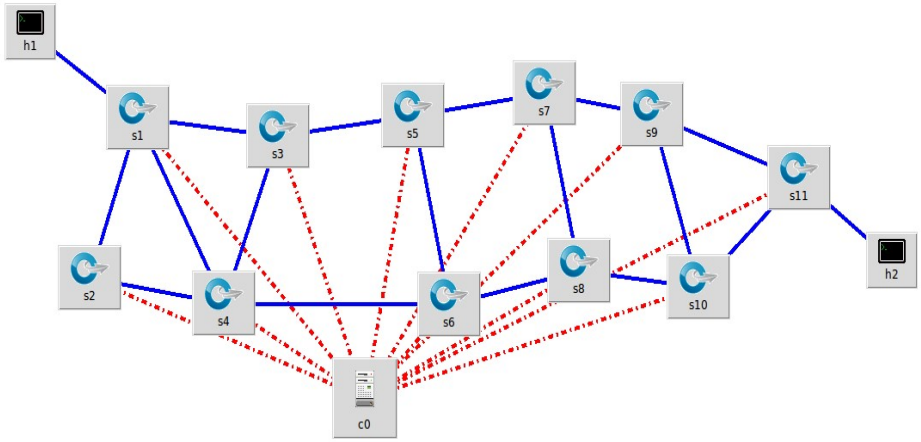
Pada bagian ini akan dijelaskan langkah-langkah untuk merancang dan mengimplementasikan *Fast-failover* pada *Multipath Routing* dengan menggunakan algoritme DFS pada *OpenFlow* versi 1.3 dengan berpedoman pada metodologi penilitan pada BAB 3 yang telah dibahas.

4.1 Perancangan

Perancangan dilakukan untuk merancang sistem yang akan dibangun menurut kebutuhan yang telah dispesifikan.

4.1.1 Topologi

Untuk dapat memanfaatkan *Fast-failover* pada jaringan m*ultipath routing*, diperlukan suatu topologi yang memiliki beberapa jalur yang redundan yang dapat dilalui suatu node jaringan. Jalur yang redundan dapat didefinisikan dengan: untuk suatu set pasangan sumber dan tujuan pada komunikasi jaringan, terdapat beberapa jalur yang dapat dilalui oleh set pasangan tersebut. Berikut gambar topologi yang digunakan:



Gambar 4. 1 Topologi Internet

Pada gambar 4.1 adalah toplogi *Internet* yang sudah dibuat pada emulator Mininet. Pada topologi tersebut terdapat 11 *Switch, 2 host* dan 1 *controller*. Pada masing-masing *link* yang menghubungkan tiap *switch* sudah berisikan besar *bandwidth* yang digunakan yaitu 1 Ghz.

4.1.2 Algoritme pencarian jalur

Pada sistem ini digunakan algoritma DFS (*Depth-First Search*) pada pencarian jalurnya. Algoritma DFS adalah metode pencarian jalur yang dilakukan dengan menggunakan struktur data *stack* yang dapat tetap menyimpan rute, *backtracking*, kemudian dapat melakukan ekspansi lagi meskipun sudah ditemukan satu jalur ke tujuan, sehingga dapat mencari seluruh jalur yang dapat dilewati untuk menuju suatu node tertentu. Secara sederhana, algoritma DFS

dapat dimodifikasi untuk melakukan pencarian seluruh jalur dari suatu node ke node lain yang dapat dijabarkan sebagai berikut:

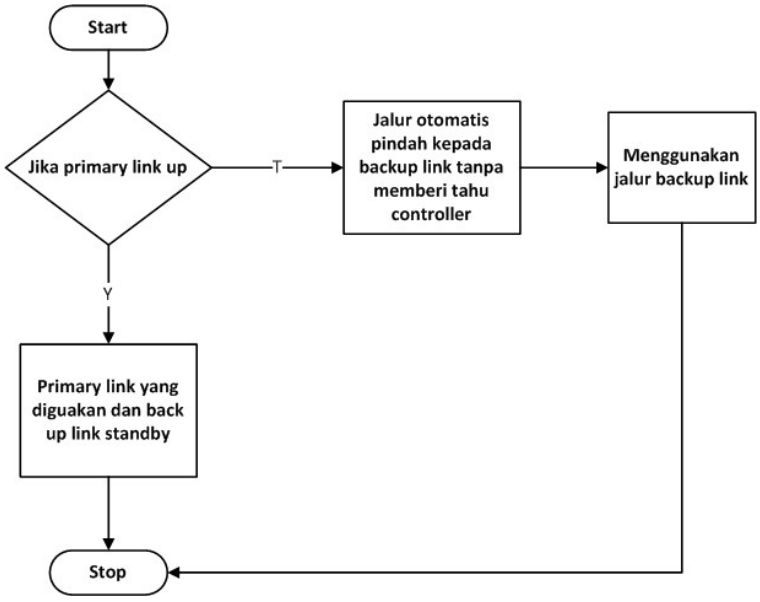
Source Code Depth-First Search

```
1   DFS(G,v) ( v is the vertex where the search starts )
2          Stack S := {}; ( start with an empty stack )
3          for each vertex u, set visited[u] := false;
4          push S, v;
5          while (S is not empty) do
6             u := pop S;
7             if (not visited[u]) then
8               visited[u] := true;
9               for each unvisited neighbour w of u
10                  push S, w;
11            end if
12         end while
13  END DFS()
```

Pada pseudocode tersebut, G menyatakan sebuah adjacency matrix dari suatu grafik, yaitu matriks yang menyimpan informasi ketetanggaan dari suatu node pada suatu grafik. Pseudocode tersebut akan menghasilkan seluruh jalur yang dapat dilalui dari node s ke tujuan d yang disimpan pada sebuah array/list P.

4.1.3 Mekanisme *fast-failover*

Berdasarkan perancangan topologi terhadap sistem yang dibangun, mekanisme *fast-failover* pada penelitian ini menggunakan fitur dari *openflow* terbaru. Mekanisme ini bertujuan untuk mengatasi permasalahan link yaitu ketika link diputus. Pada penerapan pencarian jalur yang digunakan meyediakan beberapa jalur yang mungkin di lewati dengan cara menyimpan pada *stack* yang ditampilkan pada *controller*.



Gambar 4. 2 Flowchart mekanisme *fast-failover*

Berdasarkan gambar 4.2 *flowchart* mekanisme *fast-failover*, pertama sistem akan melakukan kondisi yaitu pada *primary link* apa tidak terjadi pemutusan *link.* jika *primary link* tidak ada pemutusan *link*, *primary link* akan digunakanya dan *backup link* akan *standby* ketika *primary link* tiba-tiba ada gangguan. Jika ada gangguan pada *primary link*, *backup link* akan aktif tanpa memberitahu *controller*.

Gambar 4. 3 Flowchart otomatis mencari jalur

Berdasarkan gambar 4.3 dijelaskan bagaimana *primary link* otomatis pindah ke jalur *backup*nya ketika ada gangguan *link*. ketika belum ada pemutusan *link backup* dalam mode *standby*. Ketika *primary link* putus atau ada gangguan *backup link* langsung dinyalakan dan paket diteruksan ke jalur *backup*.

4.2 Implementasi

Berikut ini adalah tahap-tahap yang dilakukan pada implmentasi sesuai dengan metodologi penelitian, yaitu sebgaui berikut:

4.2.1 Instalasi

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

4.2.1.1 *Mininet*

Mininet adalah emulasi jaringan yang digunakan untuk mendukung pengembangan pada sistem dipenelitian ini. Berikut dijelaskan langkah-langkah intsalasi *mininet* pada *operating sistem Ubuntu*.

1. Mengunduh *source code MIninet* pada *github Mininet* dengan perintah berkut pada *Terminal:*

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

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

   $ sudo mininnet/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. Sebelum melakukan instalasi *Ryu Controller,* melakukan instalasi *python-pip* terlebih dengan menjalan perintah berikut pada terminal:

   $ sudo apt-get install python-pip

2. Untuk memulai proses installasi *Ryu Controller* dengan perintah:

   $ sudo pip install ryu

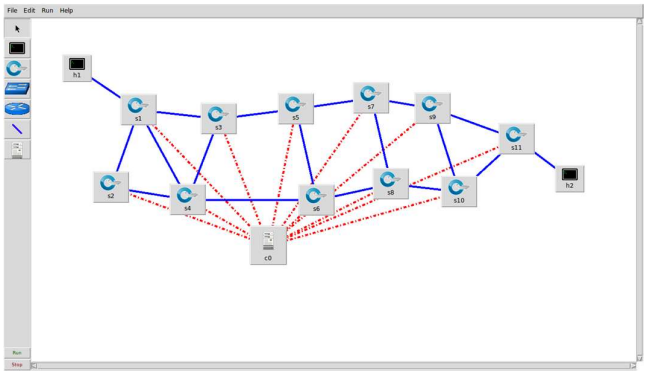
4.2.2 Membangun Topologi di *Mininet*.

Setelah melakukan intalasi perangkat lunak yang mendukung metodologi penelitian. Langkah selanjutnya adalah membangun topologi di *mininet* sesuai dengan desain topologi yang telah ditentukan. Salah satu cara untuk melakukan pengmebangan topologi di *mininet* adalah dengan menggunakan GUI yang bernama *Miniedit* yang akan dibahas dibagian ini, berikut ini langkah-angkah membangun topologi di *miniedit,* yaitu sebagai berikut.

1. Untuk menjalankan *Miniedit* jalankan perintah berikut pada terminal

   $ sudo mininet/examples/miniedit.py

2. Membangun topologi. Salah satu contoh hasil pembangunan topologi dapat dilihat pada gambar 4.4



Gambar 4. 4 Tampilan MiniEdit dan Contoh Topologi

3. Melakukan pengaturan *controller* berdasarkan tabel 4.1, 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"*.

Tabel 4. 1 Pengaturan pada pengaturan *controller*

Jumlah *controller*	1
Controller port	6633

Controller Type	*Remote Controller*
Protokol	TCP
Ip Address	127.0.0.1

4. Melakukan pengaturan pada *preferences* yang terletak pada menu *edit* untuk mengubah versi *OpenFlow* menjadi versi 1.3.
5. Untuk menjalankan simulasi dengan cara menekan tombol “run” pada sisi kiri barah pada jendela.
6. Membuka terminal baru untuk menjalankan program *controller* dengan melakukan perintah berikut:

   $ sudo ryu-manager –observe-*link*s [nama_program].py

4.2.3 Pengembangan Program *Controller*

Pengembangan program *controller* terdiri dari atas langkah-langkah pemrograman yang dilakukan *Ryu Controller* dengan menggunakan bahasa pemrograman *python* dalam mengimplementasikan logika atau kecerdasan dari berdasarkan yang telah dilakukan,

4.2.3.1 Source code Pencarian Jalur Algorima DFS

Untuk menemukan seluruh jalur antara bebrapa node pada jaringan, digunakan algoitma DFS. Berikut ini dari algoritme tersebut untuk sistem yang dibutuhkan yang ditunjukkan sebagai berikut:

Code algoritme Depth-First Search

```
...
144 def get_paths(src, dst):
145     '''
146     Get all paths from src to dst using DFS algorithm
156     '''
148     paths = []
149     stack = [(src, [src])]
150     while stack:
151         (node, path) = stack.pop()
152         for next in set(adjacency[node].keys()) - set(path):
153             if next is dst:
154                 paths.append(path + [next])
155             else:
156                 stack.append((next, path + [next]))
157    print "Available paths from ", src, " to ", dst, " : ",paths
158
```

```
...     return paths
```

dijelaskan untuk mencari jalur dengan Algoritme DFS. Dan untuk mendapatkan jalur pada code ini akan menampilkan beberapa jalur yang mungkin akan dilewati. Dan paket yang akan lewat memliki jalur yang pendek. Hasil pencarian jalur akan mengembalikan sebuah daftar jalur yang ditemukan antara dua node. Format data dari sebuah jalur adalah sebuah *list* yang berisi seluruh *switch* yang dilalui suatu jalur tersebut secara berurutan.

4.2.3.2 *Source code* Mekanisme *Fast Failover*

Untuk mengatasi permasalahan pada sebuah jalur misalnya seperti *link down* pada jaringan. Sistem membutuhkan mekanisme *fast failover* yang ada didalam *OpenFlow* versi 1.3. mekanisme *fast failover* akan ditunjukkan sebagai berikut:

Code Mekanisme *Fast-Failover*

```
...
288 for port in out_ports:
289         bucket_weight = 0
290         bucket_action = [ofp_parser.OFPActionOutput(port)]
291         buckets.append(
292             ofp_parser.OFPBucket(
293                 weight=bucket_weight,
294                 watch_port=port,
295                 watch_group=ofp.OFPG_ANY,
296                 actions=bucket_action
297             )
298         )
299 # If GROUP Was new, we send a GROUP_ADD
300 if group_new:
301 # print 'GROUP_ADD for %s from %s to %s GROUP_ID %d out_rules
302 %s' % (node, src, dst, group_id, buckets)
303    req = ofp_parser.OFPGroupMod(
304        dp, ofp.OFPGC_ADD, ofp.OFPGT_FF, group_id,
305        buckets
306    )
307    dp.send_msg(req)
308
309 # If the GROUP already existed, we send a GROUP_MOD to
310 # eventually adjust the buckets with current link
311 # utilization
312 else:
```

```
313     req = ofp_parser.OFPGroupMod(
314         dp, ofp.OFPGC_MODIFY, ofp.OFPGT_FF,
315         group_id, buckets)
316     dp.send_msg(req)
317 # print 'GROUP_MOD for %s from %s to %s GROUP_ID %d out_rules
318
319 actions = [ofp_parser.OFPActionGroup(group_id)]
320
321 self.add_flow(dp, 32768, match_ip, actions)
322 self.add_flow(dp, 1, match_arp, actions)
...
```

Pada *code* mekanisme *fast-failover* menjelaskan bagaimana mekanisme *fast-failover* akan berjalan. Pada mekanisme *fast-failover* memiliki daftar *bucket* seperti pada baris 291 yang berisikan *watch_port/group*. Selain memiliki daftar ini, Setiap *bucket* mempunyai *watch port* dan/atau *watch group* sebagai parameter khusus seperti Pada *code* mekanisme *fast-failover* pada baris 294 dan baris 295. *Watch port/group* akan memonitor "*liveness"* atau status *up/down* dari *port/group* yang ditunjukkan. Jika *liveness* dianggap down, maka *bucket* tidak akan digunakan, jika *liveness* berusaha akan *up*, maka *bucket* akan digunakan. Hanya satu *bucket* yang dapat digunakan pada satu waktu, dan *bucket* yang digunakan tidak akan berubah kecuali jika "*liveness" watch port/group* dari *bucket* berpindah dari *up* ke *down.* Saat kejadian tersebut, *fast-failover* group dengan cepat akan memilih *bucket* selanjutnya didaftar *bucket* dengan sebuah *watch port/group* yang menyala.

Pada baris 304 yang terdapat OFPGT_FF (*OpenFlow Group Table Fast Failover*). *Fast-failover group* dirancang untuk mendeteksi dan merespon untuk kegagalan *port* pada *switch*. Setiap *bucket* mempunyai *watch port*/*group* dengan parameter khusus, yang memonitor *liveness* dari *port* atau *group* yang dipantau. Hanya satu *bucket* yang digunakan pada satu waktu, dan bucket akan hanya diubah jika *watch port/group bucket* bertransisi dari *up* ke *down*. Setelah kejadian seperti itu, *bucket* yang lain akan dipilih yang mana *watch port*/*group* yang berindikasikan *link up.*