BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Pada sub bab ini dilakukan kajian terhadap penelitian-penelitian sebelumnya yang terkait dengan penelitian ini dan dijadikan sebagai pedoman dalam pelaksanaan penelitian. Berikut tabel perbandingan penelitian terdahulu dan sekarang.

No	Nama Penulis, Tahun dan judul	Persamaan	Perbedaan	
			Penelitian terdahulu	Rencana penelitian
1	Aiman Ghannami, Chen Xi Shao. 2016. Efficient Fast Recovery Mechanism in Software-Defined Networks	Menggunakan *Multipath Routing* pada *OpenFlow* Software-Defined network	Menggunakan *controller floodlight* untuk mengatur trafik pada forwading	Menggunakan *controller ryu*
2	Antonio Capone, Carmelo Cascone, Allesandro Q.T. Nguyen dan Brunilde Sanso. 2015. Detour Planning for fast and reliable Failure Recovery in SDN with OpenState	Menggunakan *Fast-failover* Pada Software-Defined Network	Diimplementasikan pada OpenState	DiImplementasikan pada beberapa jalur yang saling terhubung (*Multipath*)
3	Ying-Dar Lin, Hung-Yi Teng, Chia-Rong Hsu, Chun-Chieh Liao	Menggunakan Mulltipath Routing dengan Fast Failover pada	Metode menggunakan dua mekanisme yaitu Fast Failover dan Switchover pada	Metode hanya berfokus pada satu metode yaitu menggunakan *Fast-failover Link*.

	dan Yuan-Cheng Lai. 2016. Fast Failover and *Switch*over For *Link* Failures and Congestion Software Defined Network	Software Defined Network dan menggunakan *Ryu Controller*	arsitektur Software Defined Network	

Tabel 2. 1 Tabel perbandingan penelitian terdahulu

2.2 Dasar Teori

Pada sub bab ini, dijabarkan berbagai teori-teori yang dapat mendukung penelitian ini, yaitu berbagai teori untuk mengatasi masalah *link* pada suatu jaringan *Multipath Routing* dengan menggunakan *fast-failover link* pada *OpenFlow Softwre-Defined Network*.

2.2.1 Software-Defined Network

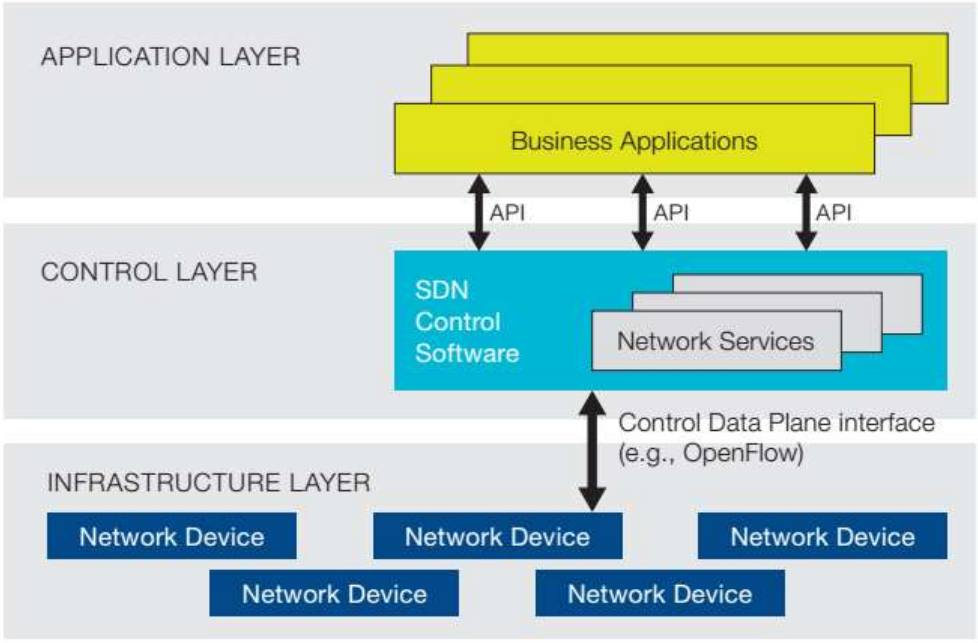
Software-Defined Network (SDN) adalah arsitektur yang muncul yang bersifat dinamis, dapat di kelola, hemat biaya dan mudah beradaptasi, sehingga ideal untuk *high-bandwidth*, sifat dinamis dari aplikasi saat ini. Arsitektur ini memisahkan *network control* dan fungsi *forwading* yang memungkinkan control jaringan langsung diprogram dan dasar infrasturktur yang abtraksi untuk aplikasi dan layanan jaringan. (Open Networking Foundation, n.d.).

Di sebuah arsitektur SDN pada perangkat *router* atau *switch* terdapat dua bagian yang disebut dengan *control plane* dan data *plane/forwading plane*. Control plane disini berfungsi untuk menjalankan logical dari sebuah perangkat. Sedangkan data *plane* merupakan bagian yang berfungsi untuk memforward paket data melalui port-port *interface* dalam perangkat jaringan

Beberapa aspek penting dari SDN adalah (Aris Cahyadi Risdianto, Muhammad Arif, & Eueung Mulyana, n.d.) :

1. Adanya pemisahan secara fisik/eksplisit antara *forwading/data-plane* dan *control-plane*.
2. Antarmuka standard (vendor-agnosic) untuk pemrograman perangkat jaringan.
3. *Control-plane* yang terpusat (secara logika) atau adanya sistem operasi jaringan yang mampu membentuk peta logika (logical map) dari seluruh jaringan dan kemudian memrepresentasikannya melalui (sejenis) API (Application Programming Interface).

4. Virtualisasi dimana beberapa sistem operasi jaringan dapat mengontrol bagian-bagian (slice atau subtrates) dari perangkat yang sama.



Gambar 2. 1 Arsitektur dari Software-Defined Network

Sumber: (Open Netwwork Foundation, 2012)

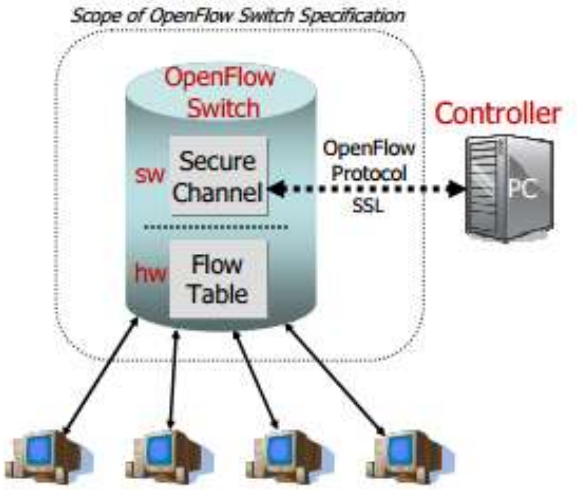
Pada Gambar 2.1 menggambarkan logika dari arsitektur SDN. Logika pada SDN terpusat pada controller SDN, yang menjaga gambaran global dari jaringan. Akibatnya jaringan ditunjukkan pada aplikasi dan menjaga perangkat menjadi satu, logical switch (Open Netwwork Foundation, 2012). Control layer pada gambar diatas memegang peran penuh untuk menganedalikan sebuah sistem pada jaringan. Salah satu contoh cara komuniksai perangkat dengan controllernya menggunakan suatu protokol, yaitu *openflow. Openflow* bisa dikatakan sebuah unit control prosessor seperti pada sebuah komputer (Road & Alto, 2012)

2.2.2 *OpenFlow*

OpenFlow adalah antarmuka komunikasi standar pertama kali didenifisikan antara control dan forwading lapisan asitektur SDN. *OpenFlow* memungkinkan akses langsung ke dan manipulasi forwading bidang perangkat jaringan seperti *switch* dan *router*, baik fisik dan virtual (berdasarkan *hypervisor*) (Open Networking Foundation, n.d.).

OpenFlow merupakan sebuah protokol terbuka untuk program *flow table* di *switch* dan *router* yang berbeda. Datapath dari sebuah *switch OpenFlow* terdiri dari sebuah *Flow Table* dan hal yang terkait dengan setiap *Flow entry*. Hal yang didukung oleh *OpenSwitch* adalah *extensible*. Tetapi ini menjelaskan syarat minimum dari semua *switch*. Untuk kinerja yang tinggi dan datapath yang rendah

biaya dengan berhati-hati yang ditentukan tingkat fleksibilitasnya. (Nick Mckeown, et al., 2008).



Gambar 2. 2 Arsitektur *OpenFlow*

Sumber: (Nick Mckeown, et al., 2008)

Menurut (Nick Mckeown, et al., 2008) berdasarkan gambar 2.2, *Switch OpenFlow* terdiri dari 3 bagian, meliputi:

1. Flow Table: dengan hal yang terkait dengan setiap flow entry, untuk memberi tahu *switch* bagaimana proses flow.
2. *Secure channel* yang terhubung dengan *switch* untuk remote control proses (disebut controller), yang memungkinkan command dan paket untuk mengirim diantara sebuah controller dan *switch* yang digunakan.
3. Protokol *OpenFlow*, yang menyediakan cara yang terbuka dan standar untuk komunikasi dengan sebuah *Switch.*

2.2.2.2 Mekanime *Fast-failover (FF)*

Mekanisme *failover* merupakan suatu teknik pada jaringan dengan memberikan dua jalur koneksi (primary dan backup link) atau lebih dimana ketika jalur utama mati, maka koneksi akan tetap berjalan dengan mengalihkan ke backup link. Prosedur mekanisme failover menentukan kelangsungan operasional jaringan. Mekanisme failover dapat dirancang sehingga dapat segera mungkin untuk bertindak ketika ada gangguan yang muncul (Purwiadi, Yahya, & Basuki, 2018)

Mekanisme *fast-failover* terdapat pada *OpenFlow* terbaru. *Fast-Failov*er ini termasuk sebuah tipe *group* dari *OpenFlow*. Mekanisme tersebut menjalankan pertama *bucket* yang hidup. *Bucket* yaitu tempat untuk menampung *group action* pada *group table*. Setiap *action bucket* dikaitkan dengan *port/group* yang tertentu untuk mengkontrol *liveness*. *Bucket* dievaluasi sesuai urutan yang ditentukan oleh

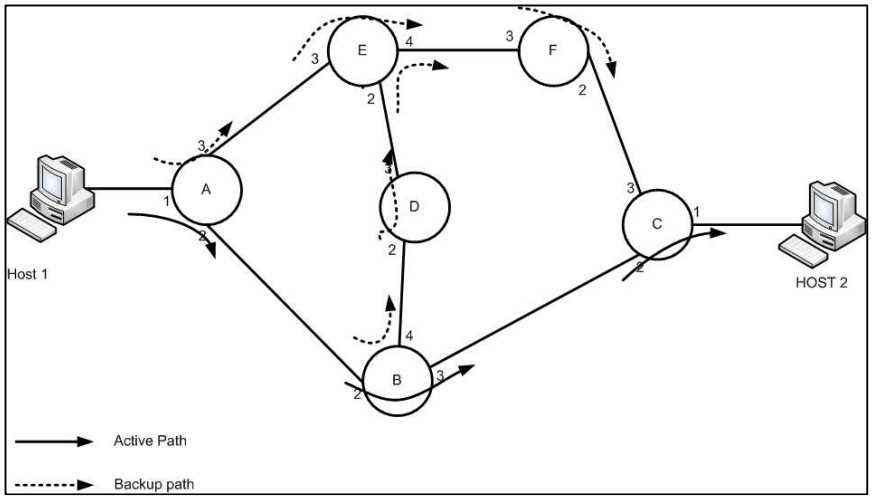
group, dan *bucket* yang pertama dikaitkan dengan sebuah *live* port/grup yang dipilih. Tipe grup ini memungkinkan *switch* untuk mengganti *forwading* tanpa memerlukan *round trip* ke *controller*. Jika tidak ada bucket yang hidup, paket akan di drop. (Open Network Foundation, 2015)

Fast-Failover Group mendukung *liveness monitoring*, untuk menentukan *bucket* yang spesifik untuk dijalankan. *Bucket group* secara explisit memantau *liveness* dari *port* atau grup yang lain. Aturan untuk menentukan liveness meliputi:

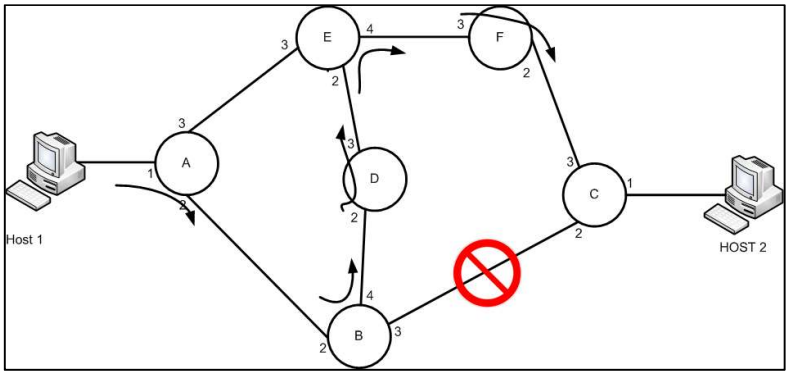
- *Port* dianggap hidup jika jika ada di datapath dan memiliki *OFPPS_LIVE* yang ditetapkan pada portnya.
- Bucket dianggap hidup jika *watch_port = not OFPP_ANY* dan *port* dipantau secara langsung saat program dijalankan atau jika *watch_group = not OFPP_ANY* dan *group* dipantau secara langsung.
- Group dianggap hidup jika bucket setidaknya satu yang hidup.

Controller dapat menyimpulkan keadaan *liveness group* dengan memantau kedaaan berbagai *port* (Open Network Foundation, 2015).

Dengan *fast-failover* group, program *controller* harus mengatisipasi setiap kegagalan yang mungkin terjadi dan menghitung kembali jalur backup yang sesuai (termasuk di luar interaksi antara bagian traffic cadangan yang berada), daripada reaksi untuk kegagalan *link* yang terjadi (Mark Reitblatt, Marco Canini, Arjun Guha, & Nate Foster, 2014).



(a)



(b)

Gambar 2. 3 a. contoh mekanisme *fast-failover b. backup link digunakan ketika ada gangguan link*

Sumber: (Ying-Dar Lin, Hung-Yi Teng, Chia-Rong Hsu, Chun-Chieh Liao, & Yuan-Cheng Lai, 2016)

Pada Gambar 2.3.a contoh ilustrasi dari prosedur *fast-failover*. Link yang aktif adalah <ABC>, kemudian *backup link*ya yaitu <AEFC> dan <ABDEFC> dan juga berpotensi ganguuan *link* pada jalur <AB> dan jalur <BC>. Pada gambar 2.3.b menunjukkan bagaimana jalur *backup* <ABDEFC> digunakan ketika ada kegagalan di jalur <BC>. Ketika ada gangguan dari *output port* 3 terdeteksi oleh *OpenFlow switch* B, paket akan diteruksan ke *OpenFlow switch* D dengan *output port* 4 (Ying-Dar Lin, Hung-Yi Teng, Chia-Rong Hsu, Chun-Chieh Liao, & Yuan-Cheng Lai, 2016).

2.2.3 Controller

Controller SDN adalah sebuah aplikasi di SDN yang mengelola *flow control* untuk mengaktifkan jaringan cerdas. *Controller* SDN didasarkan pada protokol, seperti *OpenFlow,* yang memungkinkan server untuk memberitahu *Switch* dimana untuk mengirimkan paket (Rouse, SDN Controller (Software-defined networking controller), 2012).

Controller SDN adalah inti dari komponen SDN. Komponen Ini terletak diantara diantara perangkat forwading dan aplikasi. Semua aplikasi berkomunikasi dengan perangkat melalui controller SDN. Controller SDN mendukung banyak protokol untuk mengkofigurasi perangkat jaringan. Controller memilih jalur jaringan yang cocok untuk trafik aplikasi yang berbeda. Controller SDN berfungsi sebagai sistem operasi untuk seluruh jaringan. Karena control jaringan dipindahkan dari perangkat, controller membantu untuk mengatur jaringan secara otomatis dan meudahkan integrase jaringan bisnis (Hitha Shiva & George Philip C, 2016).

2.2.4 *Ryu Controller*

Ryu dalam Bahasa jepang “flow/aliran”. Ryu adalah komponen berdasarkan SDN Framework. Ryu menyediakan komponen perangkat lunak dengan API didefinisikan dengan baik yang membuatnya mudah untuk

pengembang untuk membuat manajemen jaringan baru dan control aplikasi. Pengembangan aplikasi untuk ryu dapat dilakukan dengan menggunakan Bahasa *python* atau dengan mengirimkan pesan JSON melalui API yang tersedia. Ryu mendukung berbagai protokol untuk mengelola perangkat jaringan seperti *OpenFlow*, Netconf, OF-config dan lain-lain. Ryu mendukung penuh *OpenFlow* versi 1.0, 1.2, 1.3, 1.4, 1.5 dan Nicira Extension. Semua kode tersedia secara bebas di bawah lisensi Apache 2.0. (RYU SDN Framework Community, 2014).

2.2.5 *Multipath Routing*

Multipath adalah teknik *routing* yang menggunakan beberapa jalur alternatif melalui jaringan yang dapat menghasilkan berbagai manfaat seperti toleransi kegagalan, meningkatkan bandwidth atau meningkatkan keamanan. *Multipath Routing* telah dieksplorasi dalam beberapa konteks yang berbeda. Jaringan sirkui tradisional ada telepon menggunakan tipe dari *Multipath Routing* yang disebut *alternate path routing.* Di *alternate path routing*, tiap sumber node dan node tujuan mempunyai beberapa bagian yang terdiri dari jalur utama dan jalur alternative (Mueller, Tsang, & Ghosal, 2003).

Dalam infrastruktur *Multipath Routing*, beberapa jalur ada diantara jaringan di internetwork. *Multipath* internetwork menoleransi kegagalan ketika menggunakan dinamis routing dan beberapa protokol routing seperti OSPF, dapat menyeimbangkan beban dari traffic jaringan yang melewati beberapa jalur yang nilai metricnya sama (microsoft, 2017).

2.2.6 Algoritme *Depth-First Search* (DFS)

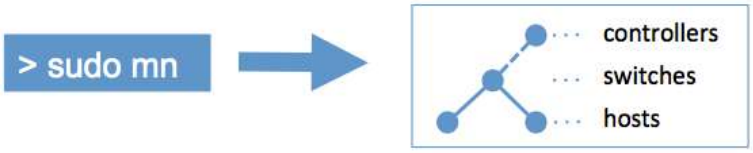
Algoritme DFS adalah salah satu algoritme yang digunakan untuk pencarian jalur (pip, 2015). DFS dilakukan dengan menggunakan struktur data stack yang dapat tetap menyimpan rute, *backtracking*, kemudian dapat melakukan ekspansi lagi meskipun sudah ditemukan jalur ke tujuan, sehingga dapat mencari seluruh jalur yang dapat dilewati untuk menuju suatu node tertentu. Pencarian dilakukan pada satu node dalam setiap level dari yang paling kiri jika pada level yang paling dalam solusi belum ditemukan, maka pencarian dilanjutkan pada node sebelah kanan. Node yang kiri dapat dihapus dari memori. Jika pada level yang paling dalam ditemukan solusi, maka pencarian dilanjutkan pada level sebelumnya. Untuk melakukan ini dengan benar algoritme ini melacak simpul yang sudah dikunjungi, sehingga kita dapat *backtrack* (Heap, 2002).

Algoritme 1: *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()
```

2.2.7 Mininet



Gambar 2. 4 Mininet Single Command

Sumber : (Mininet Team,2012)

Mininet adalah emulator jaringan yang realistis, menjalankan kernel asli, *switch* dan aplikasi code pada satu mesin (VM, cloud atau *native*), dalam hitungan detik, dengan satu perintah seperti gambar2.4 (Mininet Team, 2012). Mininet Host menjalankan standar software jaringan Linux, dan *switch* yang mendukung *OpenFlow* untuk custom routing yang fleksibel dan SDN. Jaringan Mininet menjalankan real code termasuk aplikasi jaringan Unix/Linux serta Kernel Linux asli dan tumpukan jaringan (termasuk beberapa ekstensi kernel yang mungkin tersedia, asalkan mendukung dengan jaringan *namespaces*).

2.2.8 Topologi

Topologi (dari Bahasa Yunani topos, tempat, dan logos, ilmu) merupakan cabang matematika yang bersangkutan dengan tata ruang yang tidak berubah dalam deformasi dwikontinu (yaitu ruang yang dapat ditekuk, dilipat, disusut, direntangkan dan dipilin tetapi tidak diperkenankan untuk dipotong, dirobek, ditusuk atau dilekatkan).

Topologi adalah salah satu aturan bagaimana menghubungkan komputer (node) satu sama lain secara fisik dan pola hubungan antara komponen-komponen yang berkomunikasi melalui media/ peralatan jaringan, seperti: server, workstation, hub/switch, dan pemasangan kabel (media transmisi data). Topologi fisik berkaitan dengan bentuk jaringan, seperti bagaimana memilih perangkat dan melakukan instalasi perangkat jaringan. Sedangkan topologi logika berkaitan dengan bagaimana data mengalir di dalam topologi fisik. (Gueterres, Triyono, & Nurnawati, 2014). Topologi SDN adalah Topologi yang terpusatkan pada controller. Topologi SDN pada jaringan cukup direpresentasikan secara virtual tanpa adanya *device* secara fisik. Pada SDN topologi selalu berubah, artinya jaringan semakin besar, komplek, luas, banyak, dan dinamis sehingga semakin rumit bila akan melakuan *controlling* (Erlianto, 2016).

2.2.9 Parameter Performansi

Parameter performasi merupakan parameter untuk mengukur baik atau buruknya suatu jaringan. Berikut adalah parameter yang digunakan untuk algoritme DFS dengan *fast-failover*.

2.2.9.1 *Packet Loss*

Packet Loss merupakan suatu parameter yang menggambarkan suatu kondisi yang menunjukkan jumlah total paket yang hilang. Paket yang hilang ini dapat terjadi karena *collision* dan *congestion* pada jaringan dan hal ini berpengaruh pada semua aplikasi karena retransmisi akan mengurangi efisiensi jaringan secara keseluruhan meskipun jumlah *bandwidth* cukup bersedia untuk aplikasi tersebut (Sasmita, Safriadi, & Irwansyah, 2013). Di dalam implementasi jaringan, nilai paket loss ini diharapkan mempunyai nilai yang minimum.

Tool yang pada *packet loss* yaitu *iperf*. Perintah pada terminal *host* untuk pengujian packet loss tersebut adalah:

Iperf -s -u = host yang dijadikan *server*

Iperf -c ip-tujuan -u -P jumlah-flow -t waktu = host yang dijadikan client

Pengujian packet loss ini dijalankan oleh *client-server* dengan *client* mengirimkan paket UDP ke *server* dan *server* melakukan *listen server UDP*.

2.2.9.2 *Response time*

Menurut IBM *Dictionary of Computing, Response time* adalah waktu yang berlalu diantara akhir permintaan dan awal respon pada sistem computer. Misalnya, lamanya waktu antara akhir indikasi penyelidikan dan tampilan karakter awal ketika merespon pada terminal *user* (Rouse, texhtarget.com, 2007).

Ada juga konsep *response time* yang dirasakan, yang mana waktu awal masukan user hingga akhir response. Hal ini sebenarnya mungkin karena waktu respon yang dirasakan terlalu cepat (Rouse, texhtarget.com, 2007).

Tool yang digunakan untuk mengetahui *response time* yang digunakan adalah *ping*. Perintah yang digunakan yaitu:

Ping [ip_tujuan]

Pengujian *response time* ini dilihat pada *time* saat client mengirimkan paket ICMP pada server.