BAB II
LANDASAN TEORI

2.1 Kajian Pustaka

Dalam penelitian ini, kajian pustaka berisikan tentang informasi dari sumber jurnal penelitian yang telah dilakukan. Jurnal yang digunakan adalah jurnal yang relevan sebagai acuan penelitian. Data yang diperoleh pada penelitian sebelumnya dapat digunakan sebagai pembanding penelitian yang akan dilakukan.

Pada penelitian Roni Fernando Simarmata yang berjudul "Simulasi Jaringan Software Defined Network Menggunakan Protokol Routing OSPF dan RYU Controller" membahas mengenai penerapan kinerja *routing* OSPF Pada SDN dengan menggunakan *Ryu controller*. Penelitiann ini mensimulasikan *routing OSPF* dengan menggunakan *Ryu controller* dengan simulasi menggunakan 8 buah *switch openflow*, dan dilakukan pengukuran QOS pada jaringan ini untuk melihat perbandingannya dengan jaringan konvensional dari analisa *background traffic* yang didapat jaringan SDN lebih handal dapat dilihat dari nilai *packet loss*, *jitter*, dan *throughput*nya yang lebih baik dari jaringan konvensional dikarenakan *control plane* dan *data plane* jaringan SDN ini terpisah tidak seperti pada jaringan konvensional yang masih menyatu *control plane* dan *data plane*ny [1].

Penelitian Ridha Muldina Negara yang berjudul "Analisis Simulasi Penerapan Algoritma OSPF Menggunakan *RouteFlow* pada Jaringan *Software Defined Network* (SDN)" membahas mengenai analisis dan simulasi protokol *routing* OSPF pada teknologi SDN dengan *RouteFlow* sehingga mempermudah dalam pengontrolan jaringan dengan sistem terpusat, untuk melihat apakah peroutingan OSPF dapat berjalan dengan baik di jaringan SDN, *RouteFlow* yang digunakan dalam penelitian ini adalah *RouteFlow* dengan *controller* POX dan *OpenFlow*. Pada penelitian ini parameter yang diukur adalah *time convergence* dan *Quality of Service (QoS)* menggunakan standar nilai performansi sesuai ITU.T G1010 dengan menggunakan empat topologi dengan jumlah *switch* berbeda, yaitu 4, 6, 8 dan 10 *switch*. Hasil pengujian QoS pada topologi 4, 6, 8 dan 10 *switch* dengan parameter *delay*, *jitter*, *packet loss*, dan *throughput* masih dalam rentang nilai yang ditetapkan pada ITU-T G.1010 [2].

Penelitian Abu Riza Sudiyatmoko yang berjudul "Analisis Performansi Perutingan Link State Menggunakan Algoritma Djikstra Pada Platform Software Defined Network (SDN)" pada penelitiannya dilakukan analisis apakah *protocol routing Linkstate* IS-IS dapat diterapkan pada *platform* SDN berdasarkan parameter *throughtput, delay, jitter* dan *packet loss* serta performansi perangkat *controller* dengan 3 topologi menggunakan 7 switch 7 *Host*, 9 switch 9 *Host* dan 11 switch 11 *Host***.** Hasilnya semakin besar jaringan dalam hal ini jumlah *switch* dan *Host* serta *link* pada *infrastructure layer* maka *resource* (memory) yang terpakai pada *controller* juga akan semakin besar [4].

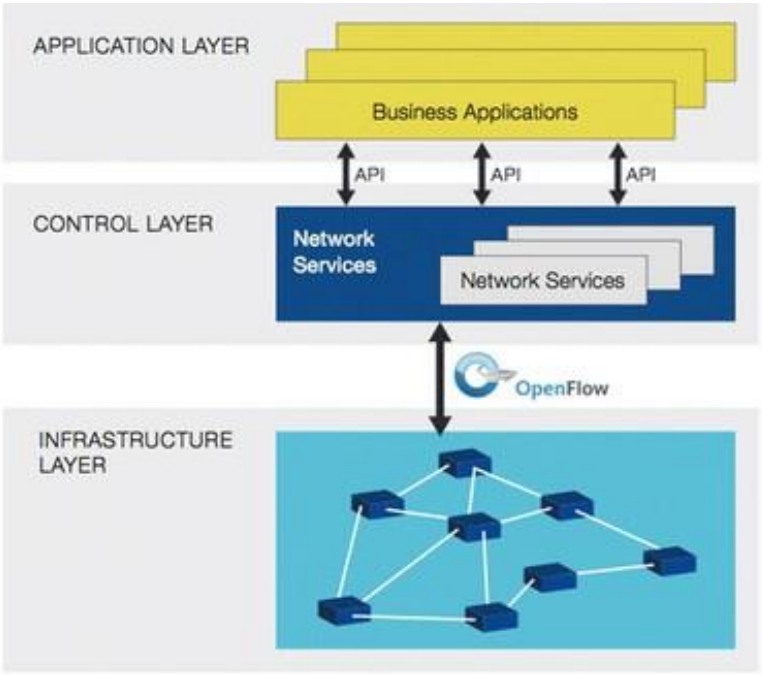
2.2 *Software Defined Network* (SDN)

SDN merupakan paradigma baru dimana pada jaringan komputer fungsi kontrol jaringan (*control plane*) juga fungsi *forwarding* data (*data plane*) dipisahkan, pemisahan ini membuat *control plane* menjadi lebih mudah dikonfigurasi. Dibandingkan dengan jaringan konvensional, pemisahan tersebut memunculkan kelebihan yaitu arsitektur jaringan lebih fleksibel, mudah dikonfigurasi, hemat biaya dan inovatif. Dalam SDN, kecerdasan jaringan secara logis terpusat pada *controller* dan perangkat jaringan menjadi perangkat *packet forwarding* sederhana yang dapat dikonfigurasi melalui *open interface* pada *data plane*. *Open interface* inilah yang disebut dengan *OpenFlow.* Sehingga seorang administrator melalui *interface* ini dapat mengontrol secara langsung lalu lintas jaringan**.**

OpenFlow ada dua yaitu *OpenFlow switch* berada pada *switch* dan *OpenFlow controller* terdapat pada *controller* SDN. *OpenFlow* adalah standar pertama mengenai antarmuka komunikasi antara *control plane* dan *forwarding layer* dalam arsitektur SDN. Sehingga O*penFlow* memungkinkan mendapatkan akses langsung dari *forwarding layer* yang berupa perangkat jaringan seperti *switch* maupun *router* atau yang bersifat *virtual*. *OpenFlow* diimplementasikan pada dua sisi yaitu dari sisi perangkat jaringan seperti *router* dan *switch* serta pada sisi *controller* SDN[4].

Gambar 2.1 menunjukkan pandangan logis dari arsitektur SDN. Arsitektur SDN membagi jaringan menjadi 3 layer yaitu *application layer, control layer* dan *insfrastructure*/data *layer*. *Application layer* merupakan *interface* terhadap seorang

admin atau peneliti dalam mengelola atau mengembangkan jaringan SDN. *Control plane* berisi suatu *controller* bersifat terpusat dan *based on software*. *Subordinate hardware* dikontrol sepenuhnya oleh *controlling plane* atau *controller* dalam melakukan *forwarding decission*. Semua *subordinate* terhubung ke *controller* [2].



Gambar 2.1 *Logical View* dari SDN Arsitektur [2]

1. *Application Layer*

Application layer berperan sebagai antar muka untuk memudahkan pengelola jaringan dalam melakukan fungsi konfigurasi, fungsi kontrol dan fungsi evaluasi. Pada *layer* ini terbetnuk dari integrasi dengan *control layer* yang memberikan informasi tentang jaringan kepada pengelola jaringan [5].

2. *Control Layer*

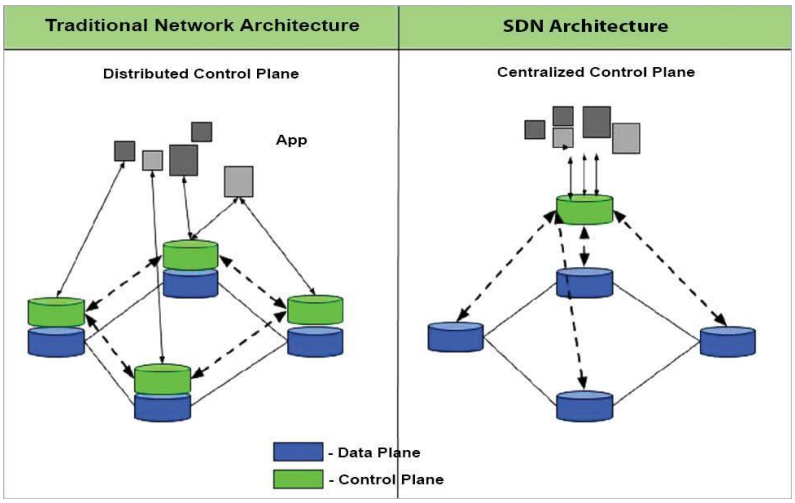
Control layer berfungsi sebagai kontrol seluruh aktifitas jaringan. Dalam *layer* ini fungsi dari *control plane* bekerja [5].

3. *Infrastructure Layer*

Infrastructure layer merupakan lapisan dasar dari arsitektur SDN. Lapisan ini berisi perangkat keras *forwarding plane* yang berfungsi sebagai *data plane*. perangkat keras ini bisa berupa *switch* atau *router* [5].

2.2.1 Perbedaan Arsitektur Jaringan SDN dan Jaringan Konvensional

Pada Gambar 2.2 mengilustrasikan perbedaan antara arsitektur jaringan SDN dan jaringan konvensional. Pada gambar di sisi kiri, menunjukan arsitektur jaringan SDN dimana control plane dan data plane dipisahkan. Sedangkan pada gambar di sisi kanan, menunjukan arsitektur jarigan konvensional dimana router memiliki fungsi control plane dan data plane berada di masing-masing router. Ini menunjukan bahwa pada jaringan SDN, control plane menggunakan struktur tersentralisasi, dan Sedangkan pada jaringan konvensional, control plane menggunakan struktur terdistribusi [1].



Gambar 2.2 perbedaan traditional *network* dan SDN *network architecture* [1]

2.2.2 *Route Flow*

RouteFlow terbentuk atas penggabungan proyek routing engine Quagga dan OpenFlow. Sistem ini terdiri dari controller OpenFlow (RFProxy), RFClient dan Independent Server (RFServer). Tujuan utama dibuatnya RouteFlow adalah menerapkan virtualisasi IP routing secara terpusat, dengan memisahkan fungsi control-plane dan data-plane. Penelitian ini memanfaatkan RouteFlow sebagai sistem yang berjalan pada controlplane. Arsitektur RouteFlow terdiri dari :

1. RFServer adalah standalone application yang mengatur kendali pusat kontrol pada jaringan. RFServer mengatur Virtual Machine (VMware) yang berjalan pada RFClient dan mengatur logic process (seperti event processing, VM mapping, resource management).

2. RFProxy adalah controller POX yang ditugaskan untuk meneruskan kebijakan pada protokol (misalnya update route, konfigurasi datapath) dari RFServer ke data-plane.
3. RFClient adalah daemon pada VMware ditugaskan untuk mendeteksi terhadap perubahan informasi routing dan memberitahukannya ke RFServer [6].

2.3 POX *Controller*

POX adalah *platform* yang digunakan untuk pengembangan dan pemodelan pada *network control software*. POX menggunakan *python* dalam bahasa pemrogramannya. POX bekerja pada layer control plan sebagai sebuah network *contoller*. Dalam arsitektur *RouteFlow POX* berada pada RFProxy yang bertanggung jawab untuk interaksi dengan *OpenFlow switch* (*datapath*) melalui protokol *OpenFlow* [7]. POX memiliki beberapa komponen yang dapat digunakan ulang untuk membuat SDN Controller sesuai dengan kebutuhan pengguna. Pada gambar 2.3 merupakan logo dari *controller* POX [8].

Gambar 2.3 *POX Controller* [7]

2.4 *Open Shortest Path First (OSPF)*

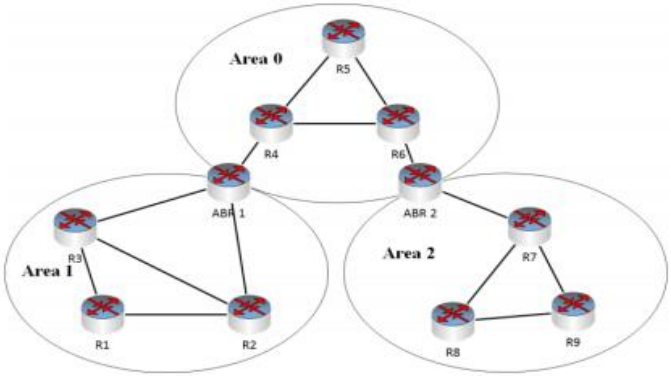
Open Shortest Path First (OSPF) merupakan protocol routing link state dan digunakan untuk menghubungkan router-router yang yang berada dalam satu Autonomus System (AS), sehingga protocol routing ini termasuk juga dalam kategori Interior Gateway Protocol (IGP) [9]. *Link state* dikembangkan menggunakan algoritma *shortest path*. Penentuan *routing* berdasarkan informasi yang diperoleh dari *router* lain. Informasi tersebut berisi kondisi aktual dari link yang terhubung dengannya. Berdasarkan informasi tersebut *router* akan memilih

"cost" terendah untuk mencapai titik tujuan. Protokol *link state* dapat mempelajari lebih banyak informasi tentang struktur jaringan, hal tersebut membuat *router* memiliki "gambaran jelas" tentang topologi jairngan dan *bandwidth* dari link topologi. LS memiliki karakteristik antara lain:

1. Dapat merespon cepat perubahan yang terjadi di jaringan;
2. Mengirimkan paket perubahan jaringan ketika ada pembaharuan;
3. Mengirimkan paket pembaharuan secara periodik pada interval tertentu disebut dengan *link state refresh.*

Proses pembaharuan *routing table* yang terjadi dalam OSPF sangat efisien, karena pembaharuan terjadi ketika ada perubahan dalam jaringan (bertambah atau berkurang). Sehingga OSPF tidak mengirimkan semua informasi yang terdapat dalam *router* ke *router* tetangga dalam 'area' tersebut. Area yang dimaksud dalam OSPF adalah sekumpulan router yang memiliki Area *ID* yang sama, diilustrasikan pada Gambar 2.4 [10].

OSPF mengharuskan administrator untuk membagi jaringan ke dalam beberapa *logical* area. Alasan mengapa harus membagi jaringan adalah untuk menjaga kinerja jaringan. Proses perhitungan *cost* pada algoritma SPF jika jaringan besar dan kompleks akan menguras *resources* dari *router*, oleh sebab itu jaringan dibagi menjadi beberapa area. Jaringan OSPF harus memiliki sebuah area khusus yang disebut dengan area *0* atau area *backbone.* Area yang lain (selain area *0*) harus terkoneksi dengan area *0* dan tidak boleh terkoneksi dengan area lain secara langsung, singkatnya semua *traffic* yang ada dalam jaringan akan melewati area *0.* Masing-masing area dikoneksikan dengan *router* yang disebut area *border routing* (ABR). ABR berfungi agar area yang lain dapat terhubung dengan area *0* [10]. Pada gambar 2.4 dapat dilihat contoh hirarki *routing* OSPF yang memiliki 3 area yaitu area 0, area 1 dan area 2.



Gambar 2.4 Contoh hirarki area OSPF [10]

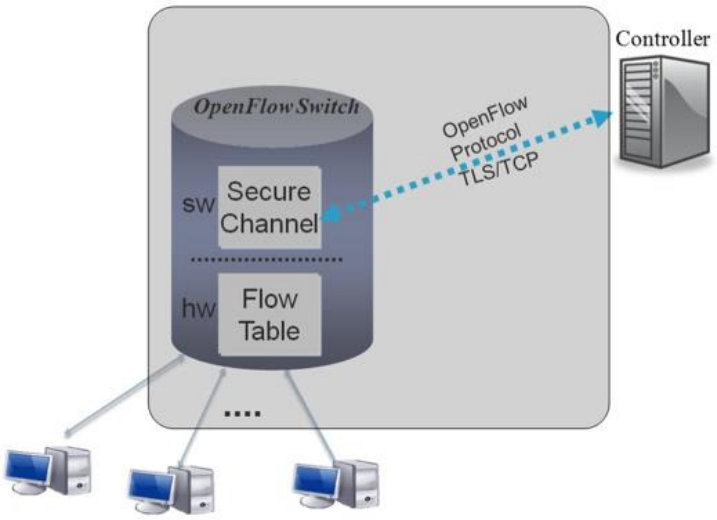
2.4.1 OSPFv3

OSPFv3 yang digunakan untuk mendukung IPv6 sesuai ketentuan RFC 5340 memiliki perbedaan utama dengan versi sebelumnya selain modifikasi *Link State Advertising* (LSA) untuk mendukung IPv6 adalah penggunaan Router-ID untuk mengidentifikasi tetangga, menggunakan alamat link lokal (Link-lokal) untuk menemukan tetangga, sehingga topologi independen dari protokol jaringan diri mereka sendiri, dan untuk memfasilitasi ekspansi di masa datang [11].

2.5 Protokol *Openflow*

OpenFlow adalah suatu protokol terbuka yang pertama kali mendefinisikan antara perangkat *control plane* yaitu *controller*, dan perangkat *data plane* yaitu *switch* pada arsitektur SDN. OpenFlow memungkinkan mengakses langsung dan manipulasi perangkat *forwarding plane* seperti *switch* dan *router*, baik fisik dan virtual =(*hypervisor-based*). OpenFlow diimplementasi dari kedua sisi antarmuka baik dari sisi infrastruktur jaringan dan kontrol *software* SDN. Sebuah OpenFlow *switch* terdiri dari *flow table* atau *grup table*. Setiap *flow table* dalam *switch* berisi satu *set flow entri*, yang terdiri dari *header field*, *counter*, dan *set of instructions* atau *actions* [12].

OpenFlow memungkinkan server memberitahukan switch jaringan tempat mengirim paket. Dalam jaringan konvensional, setiap switch memiliki perangkat lunak berpemilik yang memberitahukan apa yang harus dilakukan [13]. *Openflow* mendefinisikan infrastruktur *flow-based forwarding* dan *Application ProgrammaticInterface* (API) standart yang memungkinkan *controller* untuk mengarahkan fungsi dari *Switch* melalui saluran yang aman (*secure channel)*. Bisa dilihat pada Gambar 2.5 bahwa fungsi openflow sebagaipenghubung antara *controller* yaitu termasuk dalam *control plane* dengan *data plane* melewati *securechannel* lalu ke *flow tabel* dan diteruskan ke user [14].



Gambar 2.5 Open Flow Arsitektur [14]

2.6 Emulator *(mininet)*

Mininet adalah sebuah emulator jaringan yang membangun jaringan yang terdiri atas host virtual, switch, controller dan link. Mininet merupakan sebuah sistem untuk melakukan rapid prototyping jaringan yang sangat besar pada sumber daya yang terbatas. Mininet menggunakan virtualisasi berbasis proses untuk menjalankan banyak host dan switch pada satu kernel OS. Mininet memanfaatkan virtual software switch OpenvSwitch untuk membangun topologi OpenFlow SDN. Mininet di sini akan digunakan untuk simulasi jaringan SDN dengan berbagai macam topologi [15].

2.7 D-ITG *(Distributed Internet Traffic Generator)*

D-ITG adalah platform yang mampu menghasilkan lalu lintas data di tingkat paket secara akurat mereplikasi proses stokastik yang tepat untuk kedua IDT (inter departure time) dan ps (packet size) variable juga mendukung generasi traffic ipv4 dan ipv6 dan mampu menghasilkan lalu lintas di lapisan jaringan transportasi dan aplikasi [16].

2.8 QoS *(Quality of Service)*

QoS (Quality of Service) adalah suatu metode untuk mengetahui kualitas dari kinerja jaringan yang dirasakan secara riil. Tujuan utamanya adalah untuk

mengetahui nilai yang terdapat pada parameter QoS (*throughput, delay, jitter* dan *packet loss*) sesuai dengan nilai rekomendasi yang dikeluarkan oleh suatu lembaga seperti ETSI (*European Telecommunications Standards Institute*) yang mengeluarkan standarisasi TIPHON (*Telecommunication and Internet Protocol Harmonization Over Networks*) [17].

2.8.1 *Delay*

Delay merupakan jeda waktu yang ditempuh paket ketika dikirimkan dan diterima [18]. Besarnya *delay* dipengaruhi oleh media transmisi dan ukuran paket data. *Delay* pada saat transmisi data bisa didapatkan dari perhitungan menggunakan persamaan 2.1 [19].

$$Delay = \frac{\text{waktu penerimaan paket} - \text{waktu pengiriman paket}}{\text{jumlah paket yang diterima}} \quad \textbf{(2.1)}$$

Klasifikasi standarisasi nilai *delay* berdasarkan TIPHON (1999) diuraikan Tabel 2.1 [20].

Tabel 2.1 Klasifikasi Nilai *Delay* [20]

Kategori	**Nilai *Delay*(ms)**	**Indeks**
Sangat Bagus	<150	4
Bagus	150-300	3
Sedang	300-450	2
Jelek	>450	1

2.8.2 *Jitter*

Traffic jaringan tidak selalu senggang, terkadang *traffic* jaringan padat. Kepadatan yang terjadi menimbulkan antrian yang dibangun oleh *router*. Antrian tersebut menimbulkan penundaan yang mempengaruhi *end-to-end delay* dan terjadi variasi *delay* [21]. *Jitter* didefinisikan sebagai variasi *delay* yang diakibatkan panjang antrian dalam jaringan [18]. *Jitter* dapat diperoleh dengan perhitungan menggunakan Persamaan 2.2

$$Jitter = \frac{\text{Total Variasi } Delay}{\text{Total paket yang diterima}} \quad \textbf{(2.2)}$$

Variasi *delay* pada Persamaan 2.2 didapatkan dari perhitungan menggunakan Persamaan 2.3.

$$\text{Variasi } delay = delay - (\text{rata–rata } delay) \qquad \textbf{(2.3)}$$

Klasifikasi standarisasi nilai *jitter* berdasarkan TIPHON (1999) diuraikan Tabel 2.2 [20].

Tabel 2.2 Klasifikasi Nilai *Jitter* [20]

Kategori	**Nilai Jitter(ms)**	**Indeks**
Sangat Bagus	0	4
Bagus	0-75	3
Sedang	75 – 125	2
Jelek	125 – 225	1

2.8.3 *Throughput*

Throughput merupakan kecepatan jumlah bit data diterima yang dikirimkan melalui jaringan dari pengirim ke tujuan [18]. Jumlah *throughput* merupakan rata-rata paket data yang sukses dikirimkan oleh semua terminal pada jaringan. Persamaan 2.4 digunakan untuk mendapatkan nilai *throughput* dari suatu jaringan [19].

$$Throughtput\ (bps) = \frac{Paket\ data\ yang\ dikirim\ (bit)}{Durasi\ Pengiriman\ (\text{sec})} \qquad \textbf{(2.4)}$$

Klasifikasi standarisasi nilai *Throughput* berdasarkan TIPHON (1999) diuraikan Tabel 2.3 [22].

Tabel 2.3 Klasifikasi Nilai *Throughput* [20]

Kategori	***Throughput***	**Indeks**
Sangat Bagus	100 %	4
Bagus	75 %	3
Sedang	50 %	2
Jelek	< 25 %	1

2.8.4 *Packet Loss*

Merupakan parameter suatu kondisi yang menunjukkan jumlah total paket yang hilang, disini bisa terjadi karena collision dan congestion pada jaringan. Umumnya perangkat jaringan memiliki buffer untuk menampung data yang diterima. Jika terjadi kongesti yang cukup lama, buffer akan penuh, dan data baru tidak akan diterima [22]. Persamaan 2.5 digunakan untuk mendapatkan nilai *packet loss* dari suatu jaringan

$$Packet\ Loss = \left(\frac{Data\ yang\ dikirim - paket\ data\ yang\ diterima}{Paket\ data\ yang\ dikirim}\right) \times 100 \qquad \textbf{(2.5)}$$

Klasifikasi standarisasi nilai *Paket loss* berdasarkan TIPHON (1999) diuraikan Tabel 2.4 [22].

Tabel 2.4 Klasifikasi Nilai *Paket loss* [20]

Kategori	***Paket loss***	**Indeks**
Sangat Bagus	0 %	4
Bagus	3 %	3
Sedang	15 %	2
Jelek	25 %	1