# BAB II
# DASAR TEORI

## 2.1 KAJIAN PUSTAKA

Kajian pustaka akan membahas tentang penelitian yang telah dilakukan sesuai dengan penelitian yang akan dilakukan. Terdapat beberapa jurnal yang digunakan sebagai acuan. Penelitian Roni Fernando Simarmata , Rohmat Tulloh, dan Yuli Sun Haryani yang berjudul "Simulasi jaringan *software defined networking* menggunakan protokol *routing* OSPF dan RYU *controller"* dilakukan menggunakan aplikasi GNS3 dengan 8 *routerOS* Cisco dan 8 *host* untuk mensimulasikan jaringan konvensional , sedangkan untuk mensimulasikan jaringan *software defined network* menggunakan aplikasi Mininet dengan 8 *router* dan 8 *host* . Dari Penelitan ini dihasilkan QoS pada jaringan konvensional yaitu *throughput* sebesar 2,07 Mbit/s, *delay* sebesar 604,58 ms, *jitter* sebesar 3,17 ms, dan *packet loss* sebesar 0,007%. Sedangkan dari jaringan SDN yaitu *throughput* sebesar 2,03 Mbit/s, *delay* sebesar 0,21 ms, *jitter* sebesar 0,069 ms, dan *packet loss* sebesar 0%. [6].

Penelitian Ridha Muldina N dan Rohmat Tulloh yang berjudul "Analisis simulasi penerapan algoritma OSPF menggunakan *routerflow* pada jaringan *software defined network* (SDN)" menggunakan 4 topologi dengan jumlah *switch* berbeda dalam penelitiannya, yaitu topologi dengan 4,6,8, dan 10 *switch*. Kemudian berdasarkan parameter *convergence time* pada 4 jenis topologi didapatkan waktu sebesar 4.22 ms (untuk 4 *switch*), 4.41 ms (untuk 6 *switch*), 4.41 ms (untuk 8 *switch*) dan 4.66 ms (untuk 10 *switch*). Nilai *convergence time* yang dihasilkan menunjukan nilai yang berbeda terhadap penambahan jumlah *switch*. Penambahan jumlah *switch* mengakibatkan penambahan *convergence time* [7].

Penelitian Romi Afan dan Agus Virgono dalam penelitiannya yang berjudul "Analisis Efek Penggunaan Kontroler Ryu Dan Pox Pada Performansi Jaringan SDN" melakukan perbandingan antara *RYU controller* dan POX *controller* pada performansi jaringan SDN. Penelitian

dilakukan dengan 3 skenario topologi *full mesh* dengan 6, 8 dan 10 *switch*, dimana pada masing-masing percobaan menggunakan 2 *host* yang saling terhubung. Pada penelitian ini didapatkan hasil QoS *delay* lebih kecil dari 15 *s* untuk data, lebih kecil dari 150 ms untuk VoIP dan lebih kecil dari 10 s untuk video [8].

Penelitian Yordan Gifford Reinhart , Rohmat Tulloh , dan Hafidudin yang berjudul "Implementasi Jaringan *Software Defined Network* Dengan *Routing* OSPF dan POX sebagai *Controller"* penelitian dilakukan pada 5 perangkat *forwading plane* yang terhubung dengan laptop yang terinstal POX *controller* di dalam VMware yang berfungsi sebagai *Control Plane*. Penelitian mendapatkan hasil *throughtput* sebesar 87,98 Mbps pada protocol TCP, *delay* sebesar 0,1144 s, *jitter* sebesar 0,1607 ms, *packet loss* sebesar 0%, dan nilai *convergence time* 1,82 s untuk implementasi [9].

Penelitian Ayu Irmawati , Indrarini Dyah Irwati , dan Yuli Sun Hariyani yang berjudul "Implementasi Protokol *Routing* OSPF pada *Software Defined Network* berbasis *RouteFlow"* penelitian dilakukan dengan 4 *switch* yang terhubung dengan *control plane* sebagai pengendali sebuah jaringan. Penelitian ini mendapatkan hasil nilai *convergence time* 4,2 *s* untuk simulasi dan 4 *s* untuk implementasi. Untuk hasil simulasi didapatkan nilai *throughput* 99,8 Mbps, *delay* 27,43 *ms* , *jitter* 0,008 *ms* dan *packet loss* 0,0375%. Untuk hasil implementasi didapatkan nilai *throughput* 99,98 Mbps, *delay* 35,7 *ms* , *jitter* 1,015 *ms* dan *packet loss* 0,0956% [10].

**Tabel 2.1 Rangkuman dengan Penelitian Sebelumnya**

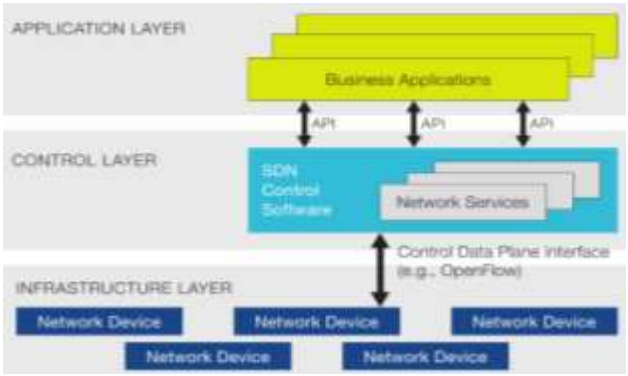
| Penelitian Oleh | Parameter Penelitian | | | |
|---|---|---|---|---|
| | *Routing OSPF* | *RYU Controller* | *POX Controller* | *Parameter QoS* |
| Roni Fernando Simarmata , Rohmat Tulloh, dan Yuli Sun Haryani | ✓ | ✓ | | ✓ |
| Ridha Muldina N dan Rohmat Tulloh | ✓ | | | ✓ |
| Romi Afan dan Agus Virgono | | ✓ | ✓ | ✓ |
| Yordan Gifford Reinhart , Rohmat Tulloh , dan Hafidudin | ✓ | | ✓ | ✓ |
| Ayu Irmawati , Indrarini Dyah Irwati , dan Yuli Sun Hariyani | ✓ | | | ✓ |
| Mayang Karmila Sari | ✓ | ✓ | ✓ | ✓ |

## 2.2 DASAR TEORI

### 2.2.1 *SOFTWARE DEFINED NETWORK*

*Software Defined Networking* (SDN) adalah sebuah konsep pendekatan jaringan dengan membuat pengontrol dan arus data dipisahkan dengan perangkat kerasnya. Awal mula terciptanya teknologi *Software Defined Networking* dimulai setelah Sun *Microsystems* merilis java pada tahun 1995. Namun pada saat itu belum cukup membangunkan para peneliti untuk mengembangkan teknologi *Software Defined Networking*. Baru pada tahun 2008 *Software Defined Networking* ini dikembangkan di UC Berkeley and Stanford University. Kemudian mulai dipromosikan pada tahun 2011 untuk memperkenalkan teknologi SDN dan *openflow*.

Karakteristik jaringan *Software Defined Networking* adalah *data plane* dan *control plane* serta menggabungkan semua *control plane* dari setiap perangkat menjadi satu *controller* yang *programmeable*. Agar seluruh *control plane* menjadi tersentralisasi sehingga mempermudah untuk mendesain dan mengoperasikan jaringan [11].




**Gambar 2. 1Arsitektur SDN [12]**

Arsitektur SDN (Software Defined Networking) terbagi menjadi 3 layer, yaitu:

1. *Application Layer*

   Berisi aplikasi *network* yang digunakan dalam sebuah perusahaan yang mencakup IDS(*Intrusion Detection System*), *load balancing* atau *firewall*. Apabila jaringan tradisional menggunakan alat tambahan untuk menambah fitur tersebut, maka pada SDN mengganti cara tersebut dengan sebuah aplikasi yang menggunakan *controller* untuk mengatur data *plane*.

2. *Control Layer*

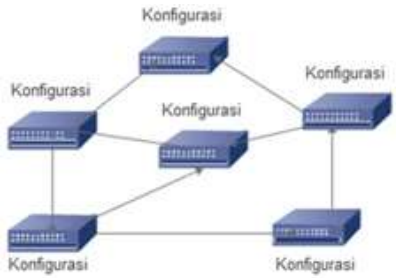
   SDN *Controller* mentranslasikan kebutuhan antara aplikasi dan infrastruktur dengan memberikan instruksi yang sesuai dengan SDN *Datapath* dan relevan dengan SDN *Application*.

3. *Infrastructure Layer*

   Mengatur SDN *Datapath* sesuai dengan instruksi yang diberikan melalui CDPI (*Control-Data-Plane Interface*).

SDN (*Software Defined Networking)* juga salah satu arsitektur baru pada jaringan yang bersifat *dynamic*, *manageable*, *cost-effective*, *dan adaptable.* Arsitektur SDN bertujuan untuk membuat jaringan menjadi lebih fleksibel dan mempermudah dalam mengontrol jaringan apabila terdapat perubahan dalam *business requirement*. Pada SDN, *network administrator* atau *network engineer* dapat membentuk lalu lintas jaringan melalui sebuah *central console*, sehingga tidak perlu mengkonfigurasi masing-masing *switch* atau perangkat yang terdapat pada topologi. Perbedaan topologi dengan SDN dan tanpa SDN:

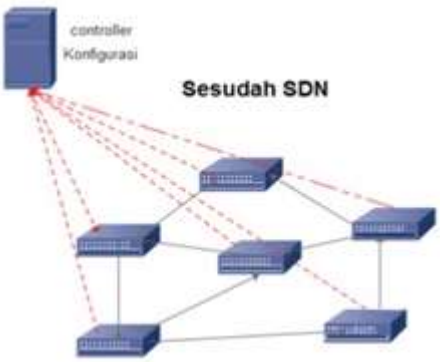
4. Topologi Tanpa SDN




https://socs.binus.ac.id/2018/12/10/software-defined-networking-sdn/

**Gambar 2. 2Topologi Tanpa SDN**

5. Topologi Dengan SDN




https://socs.binus.ac.id/2018/12/10/software-defined-networking-sdn/

**Gambar 2. 3 Topologi dengan SDN**

**Konsep *Software Defined Networking***

Konsep dasar dari *Software Defined Networking* adalah melakukan pemisahan fisik antara *Control Plane* dan *Forwading Plane*. Secara logika, *control plane* diletakan secara terpusat yang membutuhkan sebuah sistem operasi jaringan yang mampu membentuk peta logika (*logical map*) dari seluruh jaringan dan

kemudian memrepresentasikannya melalui sejenis API (*Application Programming Interface*) [13].

### 2.2.2 PENGENDALI *RYU*

https://medium.com/core-network-laboratory-tech-page/mengenal-salah-satu-controller-dalam-sdn-706ed4624660

**Gambar 2. 4 Logo aplikasi Ryu controller [14]**

RYU *controller* adalah sebuah perangkat lunak terbuka atau *open source*. RYU memiliki APIs (*application program interfac*e) sudah didefinisikan dengan sangat baik yang berarti dapat melakukan pengembangan dengan mudah untuk membuat suatu network management yang baru. yang memudahkan untuk manajemen perangkat *router* dan *switch* jaringan SDN. RYU dikembangkan dengan bahasa pemrograman python dan didukung oleh NTT (*Nippon Telegraph and Telephone*). Controller RYU ini mendukung beberapa protocol dalam software defined network diantaranya OpenFlow, Netconf, OF-config serta lainnya [15].

Sebenarnya RYU bukanlah *controller*, melainkan *framework* yang disebut *controller* dalam RYU adalah gabungan dari *framework*. RYU banyak digunakan karena mempunyai beberapa Keunggulan dibanding dengan *controller* lain diantaranya :

1. RYU menyediakan banyak komponen yang berguna untuk aplikasi software defined network.
2. Komponen lama dalam RYU dapat dimodifikasi sesuai dengan kebutuhan dan menerapkan pada komponen baru.
3. Menggabungkan komponen untuk membangun sebuah aplikasi.

*Framework* RYU berada pada *Control Layer* pada Arsitektur SDN, dan beberapa aplikasi pada RYU berada pada *Application Layer* guna untuk berkomunikasi dengan Aplikasi SDN lain menggunakan API seperti REST, RPC, dan sebagainya [16].

### 2.2.3 PENGENDALI *POX*


https://haryachyy.wordpress.com/2014/05/29/learning-pox-sdn-controller-hub-py-module/

**Gambar 2. 5 Logo aplikasi POX *controller* [17]**

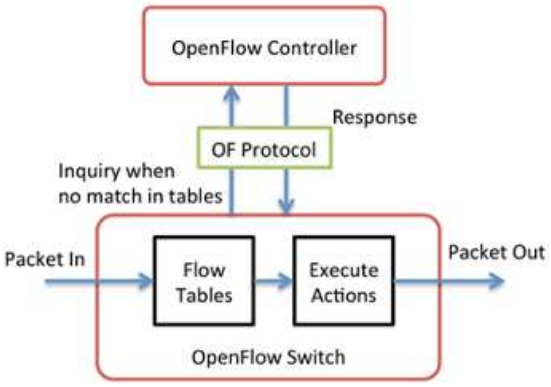
POX *controller* adalah sebuah perangkat lunak terbuka atau *open source.* POX menggunakan bahasa pemprograman python dan merupakan sebuah *platform* untuk pengembangan dan *prototyping* aplikasi jaringan yang cepat. POX diutamakan untuk suatu penelitian, POX memiliki beberapa komponen yang dapat digunakan dan digabung untuk membentuk SDN Controller sesuai kebutuhan pengguna [18]. POX *controller* menyediakan cara yang efisien untuk mengimplementasikan protokol *OpenFlow* yang merupakan protokol komunikasi antara *control plane* dan data *plane*. POX dapat menjalankan aplikasi yang berbeda seperti *hub, switch, load balancer,* dan *firewall*. Alat capture paket Tcpdump bias digunakan untuk menangkap dan melihat paket yang mengalir di antaranya POX *controller* dan perangkat *OpenFlow* [19].

### 2.2.4 *OPENFLOW*


https://www.rs-online.com/designspark/software-defined-networking-with-raspberry-pi-openflow-and-zodiac-fx

**Gambar 2. 6 Logo aplikasi *OpenFlow* [20]**

*Openflow* adalah jenis dari *Application Protocol Interfaces* (APIs) di jaringan *software defined networking* (SDN) yang digunakan untuk pengontrolan atau mengatur *traffic flows* pada *switch*. *Openflow* merupakan media komunikasi antara *control plane* dan data *plane*. *openflow* bisa digunakan pada *switch* apa saja [20]. OpenFlow sendiri adalah *southbound interface*, dimana *interface* yang memungkinkan terjadinya komunikasi antara *layer controller* dan data *plane* pada arsitektur SDN. *OpenFlow* merupakan komponen dalam arsitektur SDN yang melatar belakangi ide penelitian dalam eksplorasi dunia akademik jaringan untuk implementasi referensi standar SDN dimana teknologi ini menjadikan pergerakan momentumnya berperan cukup besar dalam industri saat ini. *OpenFlow* mempunyai 2 komponen penting yaitu *OpenFlow Controller* dan *OpenFlow Switch.*




http://www.fiber-optic-transceiver-module.com/openvswitch-vs-openflow-what-are-they-whats-their-relationship.html

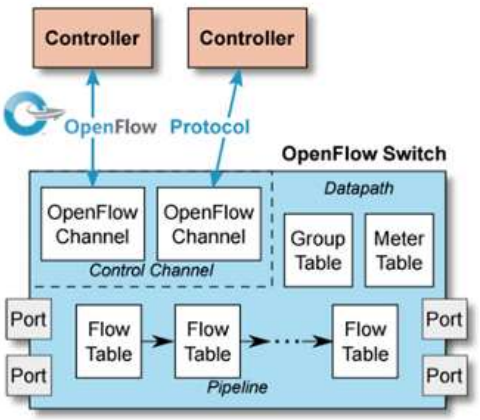
**Gambar 2. 7 Komponen pada *Openflow***

1. *OpenFlow Controller*

   *OpenFlow Controller* bertugas mengontrol *path*, memformulasikan *flow* dan mengatur kerja dari *OpenFlow switch*. Terdapat beberapa *OpenFlow controller* yang dapat digunakan seperti NOX (C *base*) , POX (*phyton base*), dan Floodlight (java *base*).

2. *OpenFlow Switch*

   Implementasi *OpenFlow Switch* dapat dilakukan dengan berbagai cara menggunakan *hardware OpenFlow switch* seperti Pronto *Switch* menggunakan *openvswitch* yang diinstall di sistem operasi berbasis linux menggunakan *ethernet card* w/ NetFPGA *support* menggunakan *Mininet OpenFlow switch emulator* sebenarnya untuk implementasi *OpenFlow* 2 komponen diatas sudah cukup, namun jika ingin dapat memonitor jaringan *OpenFlow* melalui GUI dapat dilakukan dengan menginstall ENVI dan LAVI (Nox *Base*), AVIORÂ (Floodlight *base*). Nah *OpenFlow Switch* adalah bagian yang cukup *crucial* karna akan memproses data yang datang [21].




https://overlaid.net/2017/02/15/openflow-basic-concepts-and-theory/

**Gambar 2. 8 Komponen *Openflow Switch***

Komponen openflow switch :

1. *Flow Table*
2. *Controller Channel*
3. *OpenFlow Protocol*

### 2.2.5 TOPOLOGI *FAT TREE*

Topologi jaringan *fat tree* diusulkan oleh Charles E. Leiserson pada tahun 1985. Jaringan pohon dan prosesor terhubung ke lapisan bawah. Ciri khas pohon gemuk adalah bahwa untuk setiap sakelar, jumlah tautan yang turun ke saudara kandungnya sama dengan jumlah tautan yang naik ke induknya di tingkat atas. Oleh karena itu, tautannya menjadi "lebih gemuk" di bagian atas pohon, dan sakelar di akar pohon memiliki tautan terbanyak dibandingkan sakelar lain di bawahnya [22]. Topologi *fat tree* memiliki kelebihan dalam skala jaringan besar yang dapat diterapkan pada *cloud* data *center* maupun *big enterprise*, *fat tree topology* juga dapat mendukung konektivitas antara banyak segmen jaringan yang berbeda, sehingga dapat meningkatkan *skalabilitas* dan kemanan jaringan. Dalam topologi tree, memiliki terminologi yang sama seperti *Root, parent, child* dll. Ini terutama digunakan untuk menghubungkan sejumlah besar *server* fisik / komputer di pusat data yang besar. Tujuan dari topologi *fat tree* adalah untuk menyediakan *bandwidth* yang seragam antara dua *node*. Bandingkan ini dengan model akses distribusi inti tradisional, di mana lalu lintas menjadi kelebihan permintaan pada intinya [23].

Kelebihan dari topologi *fat tree :*

1. Memiliki bandwidth yang identik di setiap bagian.
2. Setiap lapisan memiliki bandwidth gabungan yang sama.
3. Dapat dibangun dengan menggunakan perangkat murah dengan kapasitas yang seragam.
4. Setiap port mendukung kecepatan yang sama dengan *host* akhir.
5. Semua perangkat dapat mengirimkan dengan kecepatan jalur jika paket didistribusikan seragam di sepanjang jalur yang tersedia.
6. Skalabilitas hebat: sakelar k-port mendukung server k3 / 4 [24].

### 2.2.6 OSPF (*Open Shortest Path First*)

OSPF (*Open Shortest Path First*) merupakan sebuah *routing* protokol berjenis IGRP (*Interior Gateway Routing Protocol*) yang hanya dapat dijalakan dalam suatu jaringan internal suatu organisasi atau perusahaan. OSPF merupakan *routing* yang berstandar terbuka, protokol ini menggunakan konsep hirarki *routing* atau membagi-bagi jaringan menjadi beberapa tingkatan. Tingkatan-tingkatan ini diwujudkan dengan menggunakan sistem pengelompokan area dan dengan konsep ini sistem penyebaran informasinya menjadi lebih teratur, tersegmentasi dan bisa menggunakan *bandwidth* lebih efesien, lebih cepat mencapai konvergensi, dan lebih presisi dalam menentukan rute-rute terbaik menuju ke sebuah lokasi. *Routing* OSPF sangat bagus untuk diimplementasikan dalam *network* berskala besar [25]. OSPF juga sebuah protokol routing otomatis (*Dynamic Routing*) yang mampu menjaga, mengatur dan mendistribusikan informasi *routing* antar *network* mengikuti setiap perubahan jaringan secara dinamis. Pada OSPF dikenal sebuah istilah AS(*Autonomus System*) yaitu sebuah gabungan dari beberapa jaringan yang sifatnya *routing* dan memiliki kesamaan metode serta *policy* pengaturan *network*, yang semuanya dapat dikendalikan oleh *network administrator*. Dan memang kebanyakan fitur ini diguakan untuk *management* dalam skala jaringan yang sangat besar. Oleh karena itu untuk mempermudah penambahan informasi *routing* dan meminimalisir kesalahan distribusi informasi *routing*, maka OSPF bisa menjadi sebuah solusi.

Cara Kerja OSPF

1. Setiap router membuat Link State Packet (LSP)
2. LSP didistribusikan ke semua neighbour menggunakan Link State Advertisement (LSA) type 1 dan menentukan DR dan BDR dalam 1 Area.

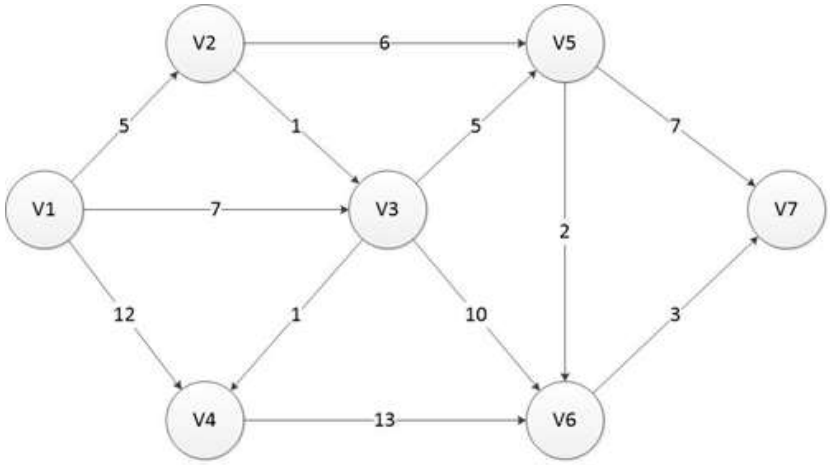
3. Masing-masing router menghitung jalur terpendek (Shortest Path) ke semua neighbour berdasarkan cost routing.
4. Jika ada perbedaan atau perubahan tabel routing, router akan mengirimkan LSP ke DR dan BDR melalui alamat multicast 224.0.0.6
5. LSP akan didistribusikan oleh DR ke router neighbour lain dalam 1 area sehingga semua router neighbour akan melakukan perhitungan ulang jalur terpendek [26].

OSPF merupakan *protokol routing* yang menggunakan *algoritma link- state* untuk membangun dan mengitung jalur terbaik ke semua tujuan yang diketahui [27] dan algoritma djikstra untuk memecahkan permasalahan jarak terpendek (*shortest path problem*) untuk sebuah graf berarah (*directed graph*). Algoritma ini dioublikasikan pada tahun 1959 jurnal Numerische Mathematik yang berjudul "*A Note on Two Problems in Connexion with Graphs" dan dianggap sebagai algoritma greedy*".

Algoritma dijkstra bekerja dengan membuat jalur ke satu simpul optimal pada setiap langkah. Jadi pada langkah ke n, setidaknya ada n node yang sudah kita tahu jalur terpendek. Langkah-langkah algoritma Dijkstra dapat dilakukan dengan langkah-langkah berikut:

1. Tentukan titik mana yang akan menjadi node awal, lalu beri bobot jarak pada node pertama ke node terdekat satu per satu, Dijkstra akan melakukan pengembangan pencarian dari satu titik ke titik lain dan ke titik selanjutnya tahap demi tahap.
2. Beri nilai bobot (jarak) untuk setiap titik ke titik lainnya, lalu set nilai 0 pada node awal dan nilai tak hingga terhadap node lain (belum terisi) 2.
3. Set semua node yang belum dilalui dan set node awal sebagai "Node keberangkatan"

4. Dari node keberangkatan, pertimbangkan node tetangga yang belum dilalui dan hitung jaraknya dari titik keberangkatan. Jika jarak ini lebih kecil dari jarak sebelumnya (yang telah terekam sebelumnya) hapus data lama, simpan ulang data jarak dengan jarak yang baru
5. Saat kita selesai mempertimbangkan setiap jarak terhadap node tetangga, tandai node yang telah dilalui sebagai "Node dilewati". Node yang dilewati tidak akan pernah di cek kembali, jarak yang disimpan adalah jarak terakhir dan yang paling minimal bobotnya.
6. Set "Node belum dilewati" dengan jarak terkecil (dari node keberangkatan) sebagai "Node Keberangkatan" selanjutnya dan ulangi langkah 5.




https://mti.binus.ac.id/2017/11/28/algoritma-dijkstra/

**Gambar 2. 9 Penentuan langkah untuk algoritma djikstra**

### 2.2.7 D-ITG

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*) variabel acak (eksponensial, seragam, *cauchy*, normal, pareto). D-ITG mendukung generasi *traffic* IPv4 dan IPv6 dan mampu menghasilkan lalu lintas di lapisan jaringan transportasi dan aplikasi. D-ITG menunjukkan properti yang lebih baik jika dibandingkan dengan *traffic* generator yang lain [28].

### 2.2.8 *QUALITY OF SERVICE*

*Quality of service* didefinisikan sebagai kemampuan untuk menjamin kebutuhan jaringan tertentu seperti *bandwidth, latency, jitter, throughput,* dan kehandalan dalam memenuhi *service level agreement* (SLA) antara provider aplikasi dan *end user*. *Switch* dan *router* bisa menandai *traffic* sehingga memungkinkan administrator jaringan untuk memprioritaskan lalu lintas tertentu, dan memberikan kemampuan untuk mendefinisikan atribut layanan yang disediakan baik secara kualitatif maupun kuantitatif [29]. .

**A. *DELAY***

*Delay* adalah total waktu untuk mengirimkan satu paket data dari pengirim ke penerima yang terdiri atas *hardware latency*, *delay* akses, dan *delay* transmisi. *Delay* yang sering dialami oleh *traffic* yang lewat adalah *delay* transmisi. Rumus menghitung *Delay* ditunjukkan pada persamaan 2.2 [30].

**Tabel 2.2 *Delay*/Latensi [30]**

| KATEGORI LATENSI | BESAR *DELAY* | *INDEKS* |
|---|---|---|
| Sangat memuaskan | < 150 ms | 4 |
| Memuaskan | 150 s/d 300 ms | 3 |
| Kurang memuaskan | 300 ms s/d 450 ms | 2 |
| Tidak Memuaskan | > 450 ms | 1 |

( Sumber : TIPHON )

Persamaan perhitungan *Delay* :

$$Delay = \frac{Packet\ Length}{Link\ bandwith} \quad (2.2)$$

## B. *PACKET LOSS*

*Packet loss* adalah parameter yang menjelaskan suatu kondisi jumlah total paket yang telah hilang selama pengiriman data karena *collision* dan *congestion.* Rumus menghitung *Packet loss* ditunjukkan pada persamaan 2.3 [30].

**Tabel 2.3 Paket *Loss* [30]**

| KATEGORI LATENSI | BESAR PAKET LOSS | *INDEKS* |
|---|---|---|
| Sangat Bagus | 0 | 4 |
| Bagus | 3 | 3 |
| Sedang | 15 | 2 |
| Jelek | 25 | 1 |

( Sumber : TIPHON )

Persamaan perhitungan *Packet Loss* :

$$Packet\ Loss = \frac{(\text{Paket data dikirim} - \text{paket diterima})\ \text{x } 100\%}{\text{Paket data yang dikirim}} \quad (2.3)$$

## C. *JITTER*

*Jitter* adalah variasi dari suatu *delay end-to-end*. Level yang tinggi pada suatu *jitter* pada aplikasi berbasis UDP adalah situasi yang tidak dapat diterima aplikasi *real-time*, seperti audio dan video. Rumus menghitung *jitter* ditunjukkan pada persamaan 2.4 [30].

**Tabel 2.4 *Jitter* [30]**

| KATEGORI LATENSI | BESAR *JITTER* | *INDEKS* |
|---|---|---|
| Sangat Bagus | 0 ms | 4 |
| Bagus | 0 ms s/d 75 ms | 3 |
| Sedang | 76 ms s/d 125 ms | 2 |
| Jelek | 126 ms s/d 225 ms | 1 |

( Sumber : TIPHON )

Persamaan perhitungan *Jitter* :

$$Jitter = \frac{\text{Total variasi } delay}{\text{Total paket yang diterima}} \quad (2.4)$$