BAB II
DASAR TEORI

2.1 Kajian Pustaka

Penulis melakukan penelitian dengan didasari oleh beberapa penelitian terdahulu tentang *load balancing* pada arsitektur SDN. Penelitian [11] mengimplementasikan metode *Round Robin* dan *OpenDayLight* sebagai *SDN controller*. Penelitian ini bertujuan untuk membandingkan perbedaan kinerja *load balancing* sebelum dan sesudah menggunakan *Round Robin* dan *OpenDayLight controller*. Penelitian dilakukan pada 19 host (3 *servers* dan 16 *clients*), 13 buah *switch* dan 1 buah *controller* dengan parameter *response time, throughput, network convergence, overhead* dan *resource utilization.* Hasil penelitian menunjukkan bahwa semua parameter yang diuji memiliki performansi lebih baik setelah diterapkan *load balancing round robin*.

Penelitian [10] menganalisis perbandingan performa algoritma R*ound Robin* dan *Least Connection* untuk *Load Balancing* pada *SDN. Pengujian* performansi menggunakan 3 kategori *rate* yaitu low, *medium*, dan *high*. Parameter pengujian yang digunakan adalah *throughput, response time*, dan *CPU Usage* menggunakan *Httperf* dan *psutil*. Hasil penelitian ini menunjukkan bahwa algoritma *round robin* lebih unggul dibandingkan algoritma *least connection* pada koneksi kecil, sementara algoritma *least connection* unggul pada koneksi besar. Keunggulan algoritma *round robin* ditunjukkan pada nilai rata-rata *response time* dan kestabilan *CPU Usage server* pada setiap kategori *rate.* Algoritma *least connection* lebih ringan beban daripada algoritma *round robin* walaupun nilainya naik di setiap kategori *rate.*

Penelitian [12] mensimulasi perancangan mekanisme *load balancing* pada *SDN*. Penelitian ini bertujuan untuk mengukur efisiensi mekanisme *load balancing* pada *SDN* dengan membandingkan penggunaan *round robin load balancing strategy* dan *random load balancing strategy*. Penelitian dilakukan dengan menggunakan 1 *POX controller,* 6 *switches,* 48 *hosts,* dengan parameter *packet loss* dan *throughput.* Hasil penelitian

menunjukkan bahwa penggunaan seleksi *round robin* dapat mengoptimasi pengurangan *packet loss*, meningkatkan *throughput,* dan memberikan kinerja yang lebih baik dibandingkan dengan teknik seleksi secara acak. Penggunaan *round robin* membantu menghindari *overload* pada sumber daya tunggal yang meminimalkan konsumsi sumber daya, kemacetan, dan mengarah pada efektivitas keseimbangan beban sehingga meningkatkan kinerja jaringan.

Penelitian [5] melakukan survei penerapan solusi dan strategi optimasi pada *hybrid SDN* dengan membandingkan berbagai strategi optimasi dari perspektif rekayasa *traffic*, penghematan sumber daya, kapasitas kontrol jaringan, dan keamanan jaringan. Penelitian ini menjelaskan bahwa *hybrid SDN* memiliki kemudahan dalam migrasi dari jaringan tradisional menuju *pure SDN*. Penerapan *hybrid SDN* menggunakan *switch* yang berperan sebagai *middlebox* untuk mengirim konfigurasi dan informasi ke seluruh jaringan. Penerapan ini hampir mendekati performa *pure SDN* walaupun hanya menggunakan 1 *switch SDN*.

Penelitian [13] bertujuan mengembangkan skema *load balancing* pada *hybrid SDN* dengan set minimal SDN *device* (1 buah *controller* dan 1 buah *switch*). Penelitian ini mengusulkan skema *load balancing* baru untuk memantau indikator beban *server* dengan menerapkan *metric multi-parameter* (CPU *load*, I/O *Read*, I/O *Write*, *Link Upload*, *Link Download*). Penerapan *metric multi-parameter* dilakukan untuk menjadwalkan koneksi guna menyeimbangkan beban di *server* seefisien mungkin. Hasil yang diperoleh menunjukkan bahwa mekanisme *load balancing* ini mencapai hasil lebih baik daripada skema *load balancing* yang ada pada jaringan tradisional dan *pure SDN*.

Penelitian [14] mengevaluasi performansi ONOS dan *OpenDayLight* sebagai *SDN controllers* dengan menggunakan emulasi Mininet dan D-ITG *traffic generator* berdasarkan parameter *delay, jitter* dan *packet loss.* Hasil eksperimen menunjukkan bahwa ONOS secara signifikan memiliki nilai yang lebih tinggi pada ketiga parameter dibandingkan dengan OpenDayLight di semua topologi. Penelitian ini menjelaskan bahwa

performansi OpenDayLight yang buruk dapat terjadi akibat pemanfaatan *CPU* secara berlebihan. Oleh karena itu, peneliti menyarankan untuk menggunakan *Hyper-threading* supaya kinerja *CPU* meningkat.

Tabel 2. 1 Rangkuman keterkaitan dengan penelitian sebelumnya

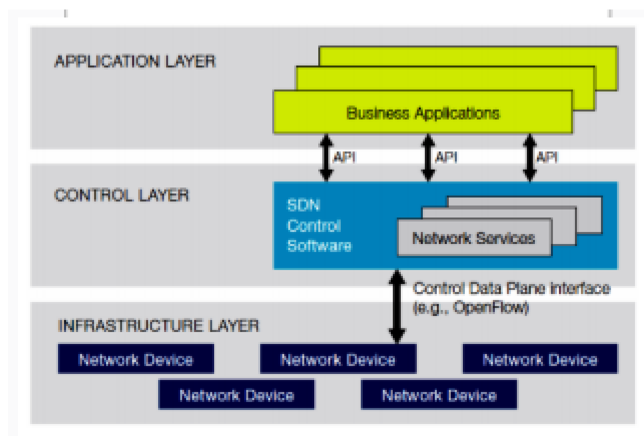
Penelitian Oleh	Parameter Penelitian						
	Routing	Round Robin	Quality of Service	Arsitektur		OpenvSwitch	ONOS
	OSPF			Hybrid SDN	Pure SDN		
Edgar Giovanni,Nachwan Mufti dan Ridha Muldina		✓	✓		✓	✓	
Agung Nugroho, Widhi Yahya, dan Kasyful Amron		✓	✓		✓	✓	
Kiran A Jadhav, Mohammed Moin Mulla, Narayan D. G		✓	✓		✓	✓	
Teodor Malbašić, Petar D. Bojović, Živko Bojović, Jelena Šuh, Dušan Vujošević			✓	✓	✓		
Xinli Huang, Shang Cheng, Kun Cao, Peijin Cong, Tongquan Wei, Shiyan Hu				✓			

Lusani Mamushiane, Themba Shozi			✓		✓		✓
Rizqi Mulya Iskandar	✓	✓	✓	✓	✓	✓	✓

Paparan kajian pustaka di atas menunjukkan bahwa penelitian sebelumnya hanya fokus pada salah satu arsitektur *SDN (pure atau hybrid SDN)*. Selain itu, hanya beberapa penelitian menggunakan *openvswitch*. Penggunaan *ONOS* sebagai *controller SDN* belum banyak dieksplorasi. Penelitian ini berupaya untuk mengisi celah penelitian sebelumnya dengan mengimplementasikan protokol routing *OSPF* pada *hybrid SDN* dan *pure SDN* menggunakan *load balancing round robin*. Penelitian ini menggunakan *OpenFlow* sebagai protokolnya dan mengukur parameter *QoS* seperti *jitter, delay, throughput*, dan *packet loss*.

2.2. Dasar Teori

2.2.1 Software Defined Network

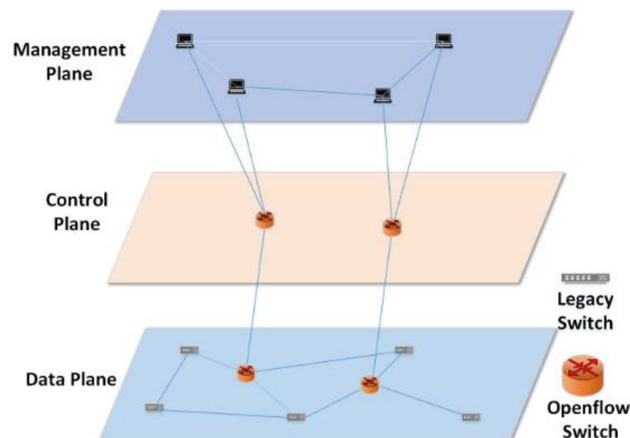


Gambar 2. 1 Arsitektur *Software Defined Network* [15]

Software defined network merupakan sebuah pendekatan baru untuk membangun, mendesain serta mengatur jaringan komputer. *Open Networking Foundation* (organisasi pengembang *SDN*) mendefinisikan *SDN* adalah suatu arsitektur jaringan dimana *control network* dipisahkan dari *system forwarding*nya. *Controller* tersebut dapat kita program secara langsung [11]. Hal ini menghasilkan arsitektur tiga tingkat seperti yang ditunjukkan pada gambar 2.1 di mana tingkat terbawah adalah *data layer*,

lapisan tengah adalah *control plane* dan lapisan paling atas adalah *application layer*. Antara ini tingkatan adalah lapisan abstraksi yaitu, *North-Bond Interface* (NBI) dan *South-Bond Interface* (SBI). NBI digunakan oleh aplikasi seperti *OpenStack*, load balancing, dan deteksi intrusi untuk menjalankan kebijakan tingkat tinggi pada perangkat keras yang diatur melalui *controller*. Berdasarkan kebijakan ini, *controller* menggunakan SBI untuk memprogram penerusan data untuk kemudi lalu lintas yang optimal. Beberapa SBI yang didukung oleh SDN pengontrol termasuk namun tidak terbatas pada, *OpenFlow, PCEP, LISP, BGP-LS*, dll [14].

2.2.2 Hybrid SDN

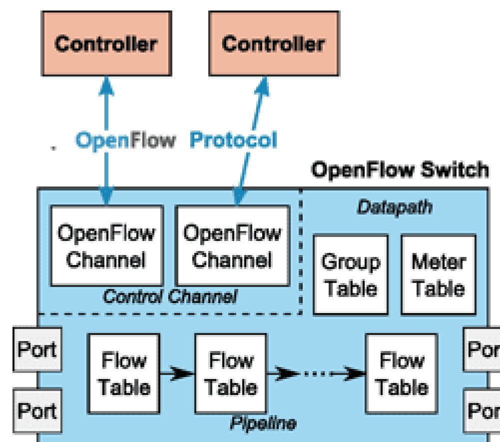


Gambar 2. 2 Arsitektur *Hybrid SDN* [16]

Hybrid SDN adalah proses transisi dari jaringan tradisional ke *SDN* yang menggabungkan kontrol pusat terprogram dengan distribusi *routing*. *Operator* dapat menyeimbangkan beban dan mengatur jaringan lebih mudah. *Controller* dalam jaringan *SDN* hanya dapat difokuskan pada arus atau lalu lintas data. Sementara sebagian besar paket dikelola oleh protokol tradisional. *Operator* hanya perlu melakukan migrasi perangkat akses lama ke *switch SDN*[16]. Dalam jaringan *hybrid SDN* yang ditunjukkan pada gambar 2.2*,* lalu lintas diteruskan berdasarkan keputusan mekanisme *switching* dan *routing* terdistribusi, kontrol lalu lintas di *hybrid SDN* tergantung pada berbagai strategi penyebaran seperti (*Island-based*, *service-based*, *class-based*, *controller-based*, *Hal-based*, *Agent-based* dan *overlay-based*) dalam hal ini, kontrol lalu lintas akan diteruskan melalui SDN dan sejumlah lalu lintas akan dikendalikan melalui SDN, lalu lintas yang lain

dikendalikan dengan mekanisme tradisional, dan pilihan tentang cara meneruskan dan memproses lalu lintas jaringan diambil oleh pengontrol yang terpusat secara logis. Dalam *hybrid SDN* dan *legacy network* masih menggunakan protokol tradisional seperti *ECMP (Equal Cost Multipath)*, *Open Shortest Path First (OSPF)* untuk meneruskan paket, hanya perangkat *Software Define Network* yang akan dikendalikan oleh pengontrol terpusat. Hal ini yang disebut dengan *hybrid SDN*. *SDN* sendiri lebih menarik untuk digunakan meskipun *hybrid SDN* memiliki perbedaannya sendiri dan tantangan yang tidak ada di *Pure SDN* [6].

2.2.3. Openflow

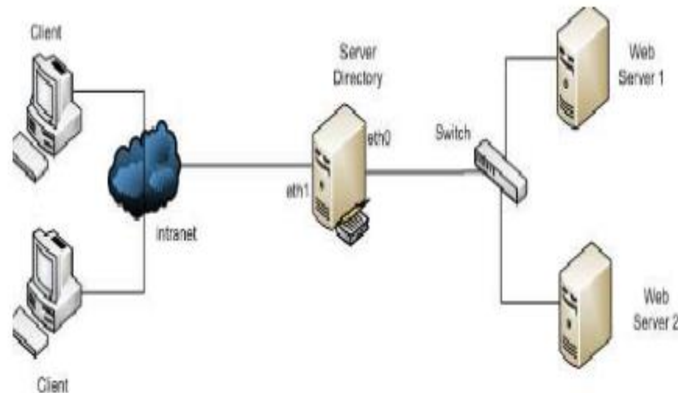


Gambar 2. 3 *OpenFlow*

Openflow adalah protokol yang meneruskan paket berdasarkan aturan yang tersimpan pada *flow table* dan mengatur keperluan komunikasi antara SDN *controller* dan *SDN switches* seperti yang ditunjukkan pada gambar 2.3. *Open flow* dapat berperan sebagai *load balancer* dan mengumpulkan data–- data yang diperlukan dan kemudian memilih *server* tujuan[17]. *Open flow* merupakan sebuah protokol perutean berjenis *IGP (Interior Gateway Protocol)* yang hanya dapat bekerja dalam jaringan internal suatu organisasi atau perusahaan. Jaringan internal maksudnya adalah jaringan di mana pengguna masih memiliki hak untuk menggunakan, mengatur, dan memodifikasinya, atau dengan kata lain, pengguna masih memiliki hak administrasi terhadap jaringan tersebut. Jika pengguna sudah tidak memiliki hak untuk menggunakan dan mengaturnya, maka jaringan tersebut dapat dikategorikan sebagai jaringan eksternal. Selain itu, *OSPF*

juga merupakan protokol perutean yang berstandar terbuka. Maksudnya, protokol perutean ini bukan ciptaan dari vendor mana pun, sehingga siapapun dapat menggunakannya, perangkat mana pun dapat cocok dengannya, dan di mana pun protokol perutean ini dapat diimplementasikan[15].

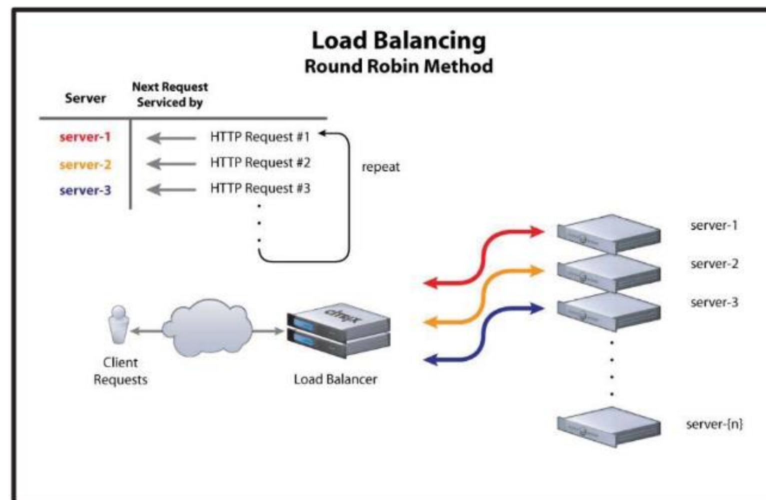
2.2.4. Load Balancing



Gambar 2. 4 Topologi *Load Balancing* [18]

Load balancing adalah suatu tehnik yang digunakan untuk memisahkan antara dua atau banyak *network* jalur. Dengan mempunyai banyak jalur maka optimalisasi utilisasi *throughput*, sumber daya, atau *response time* akan semakin baik karena mempunyai lebih dari satu jalur yang bisa saling mem-*backup* pada saat terjadi *overload* dan lebih cepat pada saat jaringan normal seperti yang ditunjukkan pada gambar 2.4. *Load balancing* sangat dibutuhkan jika diperlukan realibilitas tinggi yang memerlukan 100% koneksi *uptime* dan koneksi *upstream* yang berbeda dan dibuat saling mem-*backup.*

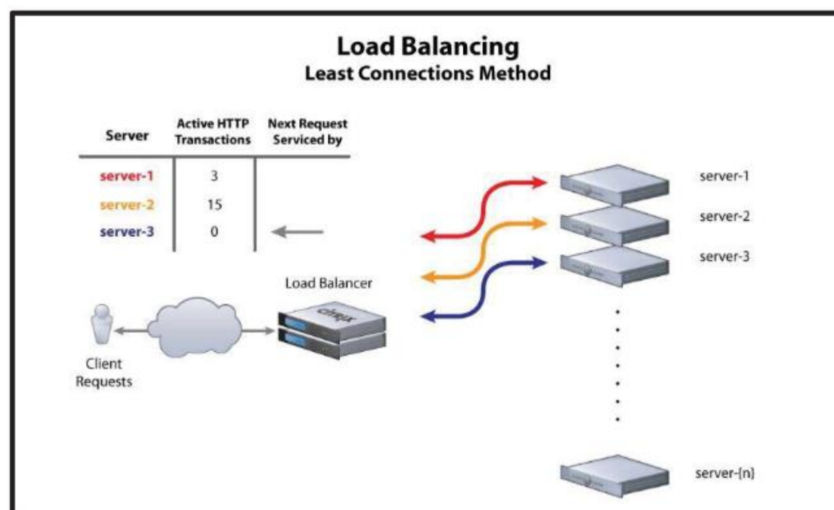
2.2.5. Algoritma Round Robin



Gambar 2. 5 Algoritma *round robin* [19]

Pada gambar 2.5, ditunjukkan algoritma *round robin* yang merupakan algoritma paling sederhana dan paling banyak digunakan oleh perangkat *load balancing*. Algoritma *round robin* bekerja dengan cara membagi beban secara berurutan dan bergiliran dari satu *server* ke *server* lainnya. Konsep dasar dari algoritma *round robin* ini adalah dengan menggunakan *time sharing*, karena algoritma ini memproses antrian secara bergiliran [19].

2.2.6. Algoritma Least Connection



Gambar 2. 6 Algoritma *least connection*

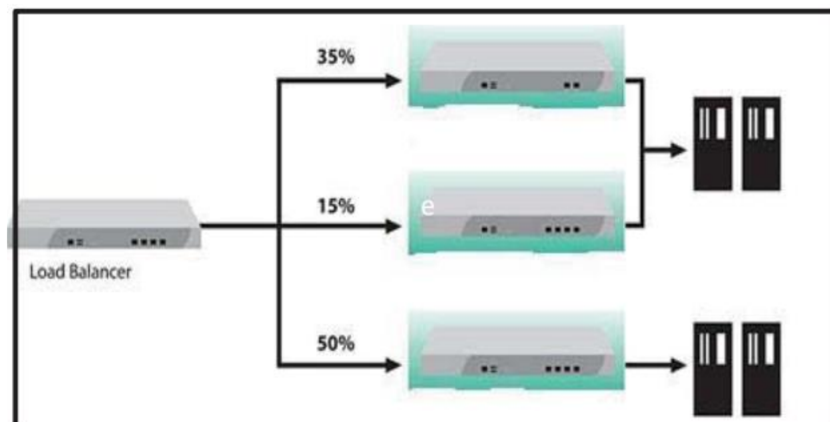
Pada gambar 2.6, ditunjukkan algoritma *least connection* dimana algoritma ini melakukan pembagian beban traffic berdasarkan pada

banyaknya koneksi yang sedang dilayani oleh *server*. *Server* dengan minim koneksi akan diberikan beban berikutnya, dan juga apabila *server* dengan koneksi sibuk akan dialihkan bebanya kepada *server* yang koneksi lebih rendah.

Algoritma ini termasuk salah satu algoritma penjadwalan dinamis, karena memerlukan perhitungan koneksi aktif untuk masing-masing real *server* secara dinamik. Mekanisme algoritma ini apabila merujuk gambar 3.1. Terdapat 3 *server*, contoh terdapat dua service-HTTP dari 3 *server* tersebut. Service HTTP tersebut berada di *server* 1 dan *server* 2. Apabila client request dan proses request masih meminta ke *server* 2 akan dialihkan ke *server* 1 dikarenakan *server* 2 masih dalam proses[10].

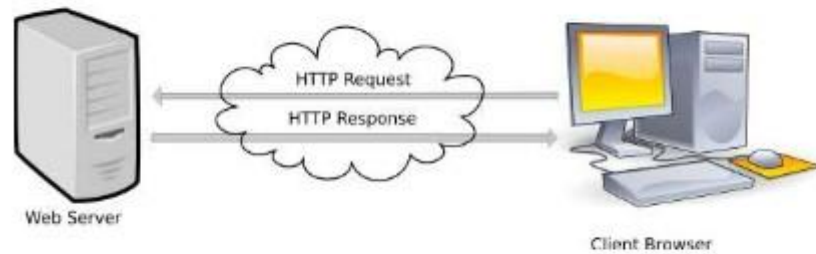
2.2.7. Algoritma Rasio

Algoritma rasio menggunakan parameter rasio pada masing-masing *server* di dalam sistem load balancing. Parameter rasio ini akan menjadi landasan pembagian beban pada *server-server* yang terlibat. *Server* dengan rasio terbesar diberi beban besar, sebaliknya *server* rasio kecil akan lebih sedikit diberi beban seperti yang ditunjukkan oleh gambar 2.7



Gambar 2. 7 Algoritma Rasio

2.2.8. Web *Server*



Gambar 2. 8 *Web Server* [20]

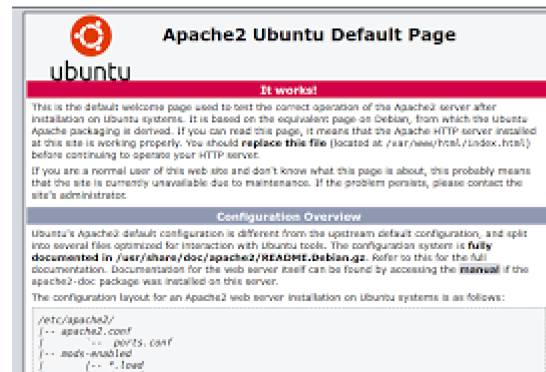
Web server merupakan perangkat lunak yang melayani request *HTTP* baik dari *client* maupun dari *server* dan *web browser*. Kemudian, web *server* akan mengirimkan kode-kode dinamis ke *server* aplikasi. *Server* aplikasi inilah yang menerjemahkan dan memproses kode-kode dinamis menjadi kode-kode statis *HTML* dalam suatu halaman statis yang kemudian dikirimkan ke *browser* oleh *web server* seperti yang ditunjukkan pada gambar 2.8. *Web server* biasanya disebut juga dengan *HTTP server*, karena basisnya menggunakan *protocol HTTP*.

2.2.9. Wireshark

Gambar 2. 9 *Wireshark*

Wireshark merupakan salah satu tools atau aplikasi untuk menangkap paket data berbasis *open-source* untuk melakukan analisis dan pemecah masalah jaringan. Selain itu juga digunakan untuk pengujian software karena mampu membaca konten dari tiap paket trafik data. Analisis kerja jaringan melingkupi berbagai hal, dimulai dari proses menangkap paket data atau informasi yang lewat di dalam jaringan sampai memperoleh informasi penting seperti *password, email* dan lain sebagainya.

2.2.10. Apache 2



Gambar 2. 10 *Apache2*

Apache Web Server merupakan *unix-based web server*, *Apache* awalnya dikembangkan dengan berbasis kode pada *NCSA HTTPS 1.3* yang kemudian diprogram ulang menjadi sebuah *web server* yang paling banyak digunakan saat ini. *Apache* kini menjadi *web server* paling populer dan 42% *domain website* yang ada di internet menggunakannya. *Apache* memiliki fitur yang sangat lengkap mulai dari performa yang baik, fungsionalitas, efisiensi, serta kecepatan yang tinggi. *Apache* juga merupakan *web server* berbasis *open source* [10]. Gambar 2.10 merupakan tampilan halaman muka *web server apache2.*

2.2.11. Quality of Services

QoS adalah kemampuan suatu jaringan untuk menyediakan layanan yang baik dengan menyediakan *bandwidth*, mengatasi permasalahan *jitter* dan *delay*. *QoS* memiliki berbagai parameter seperti *latency*, *jitter*, *packet loss, throughput, MOS*. Kualitas jaringan yang digunakan sangat menentukan *QoS*. Semakin baik jaringan,maka *QoS* akan semakin baik. Berdasarkan standardisasi TIPHON, *QoS* dinyatakan baik ketika nilai *throughput* lebih dari 1200 Kbps, *delay* 150 – 300 ms, *jitter* 75 ms, dan *packet loss* sebesar 3 – 14%. Nilai *QoS* dapat menurun karena beberapa faktor, seperti: redaman, distorsi, dan *noise* [21].

Berikut ini merupakan parameter *QoS* berdasarkan standarisasi TIPHON :

1. Throughput

   Throughput adalah kecepatan transfer data secara efektif dan memiliki satuan *bps*. Troughput merupakan jumlah total kedatangan

paket sukses yang diamati pada tujuan selama interval waktu tertentu dibagi oleh durasi interval waktu tersebut. Nilai *throughput* dapat dihitung menggunakan persamaan [22].

Persamaan 2. 1 *Throughput*

$$throughput\ (bps) = \frac{Jumlah\ data\ yang\ dikirim}{Waktu\ pengiriman\ data} = \qquad (2.1)$$

Tabel 2. 2 Standardisasi TIPHON *Throughput*

Kategori *Throughput*	***Throughput***	**Indeks**
Sangat Buruk	0 – 338 kbps	0
Buruk	338 – 700 kbps	1
Sedang	700 – 1200 kbps	2
Baik	1200 kbps – 2,1 Mbps	3
Sangat Baik	>2,1 Mbps	4

2. Delay

Delay adalah waktu tunda suatu paket yang diakibatkan oleh proses transmisi dari satu titik asal ke titik lain yang menjadi tujuannya. Untuk delay sendiri memiliki persamaan seperti dibawah ini dan memiliki satuan *s* [22].

Persamaan 2. 2 Persamaan *Delay*

$$Delay\ (ms) = \frac{Total\ Delay}{Total\ Paket\ yang\ diterima}$$

Tabel 2. 3 Standardisasi TIPHON *Delay*

Kategori *Delay*	***Delay***	**Indeks**
Buruk	> 450 ms	1
Sedang	300 – 450 ms	2
Baik	150 – 300 ms	3
Sangat Baik	< 150 ms	4

3. Packet Loss

Packet loss didefinisikan sebagai kegagalan transmisi paket IP mencapai tujuannya.*Packet loss* sendiri memiliki satuan %. Ada beberapa faktor kegagalan paket untuk mencapai tujuan, diantaranya :

a. Terjadinya kelebihan trafik didalam jaringan.

b. Tabrakan (*congestion*) dalam jaringan.
c. *Error* yang terjadi pada media fisik.
d. Kegagalan yang terjadi pada sisi penerima bisa disebabkan karena *overflow* yang terjadi pada *buffer* [22].

Persamaan 2. 3 Persamaan *Packet Loss*

$$Packet\ Loss\ (\%) = \left(\frac{data\ yang\ dikirim - paket\ data\ yang\ diterima}{paket\ \ data\ yang\ dikirim}\right) x\ 100\%$$

Tabel 2. 4 Standardisasi TIPHON Packet Loss

Kategori *Packet Loss*	***Packet Loss***	**Indeks**
Buruk	> 25%	1
Sedang	12 - 24%	2
Baik	3 – 14%	3
Sangat Baik	0 – 2%	4

4. Jitter

Jitter merupakan variasi *delay* antar paket yang terjadi pada jaringan IP dan memiliki satuan *ms*. Besarnya nilai *jitter* akan sangat dipengaruhi oleh variasi beban trafik dan besarnya tumbukan antar paket (*congestion*) yang ada dalam jaringan IP. Semakin besar beban trafik di dalam jaringan akan menyebabkan semakin besar pula peluang terjadinya *congestion* dengan demikian nilai *jitter* akan semakin besar. Semakin besar nilai *jitter* akan mengakibatkan nilai *QoS* akan semakin turun. Untuk mendapatkan nilai *QoS* jaringan yang baik, nilai *jitter* harus dijaga seminimum mungkin [22].

Persamaan 2. 4 Persamaan *Jitter*

$$Jitter\ (ms) = \frac{Total\ variasi\ delay}{Total\ paket\ yang\ diterima - 1} =$$

Total variasi *delay* didapat dengan rumus :

$$Total\ variasi\ delay = (delay2 - delay1) + (delay3 - delay2) + \cdots (delay\ n - delay(n-1))$$

Tabel 2. 5 Standardisasi TIPHON *Jitter*

Kategori *Jitter*	**Besar *Jitter***	**Indeks**
Buruk	225 ms	1
Sedang	125 ms	2
Baik	75 ms	3
Sangat Baik	0 ms	4

2.2.12. ONOS Controller

Gambar 2. 11 *ONOS Controller* [4]

ONOS sering disebut dengan komponen dasar perangkat lunak yang bersifat *open source* pada kerangka kerja jaringan. *ONOS* diimplementasikan sepenuhnya dengan oleh bahasa *Python*, dan didukung oleh laboratorium NTT. Seperti *controller software defined network* lainnya, *ONOS* juga menyediakan komponen dengan *API* yang tak tersembunyi untuk para pengembang membuat pengelolaan jaringan baru [23].

2.2.13. Protocol Routing OSPF

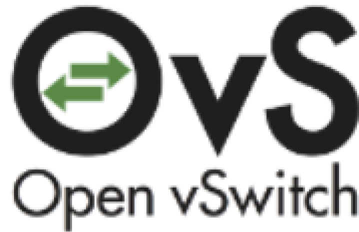
Sebuah *OSPF* bekerja berdasarkan algoritma *Shortest Path First* yang dikembangkan berdasarkan algoritma *Djikstra*. Sebagai *Interior Gateway Protocol* (*IGP*). *Interior Gateway protocol* atau *Interior Routing* Protokol dikembangkan untuk menghubungkan *router - router* dibawah kendali *administrator* jaringan [15].

2.2.14. Load Balancer HAProxy

HAProxy adalah aplikasi *open-source* yang bisa digunakan untuk kebutuhan *load balancing*. Selain dapat melakukan *load balancing, haproxy*

juga dapat mengatasi *fail over* akibat salah satu *server* tidak bisa diakses. *HAProxy* didukung oleh beberapa algoritma [24].

2.2.15. Openvswitch



Gambar 2. 12 *Openvswitch*

Openvswitch merupakan sebuah aplikasi yang digunakan untuk membangun sebuah *virtual switch* pada sistem operasi *Linux*. *Openvswitch* juga dapat diimplementasikan pada komputer desktop pada umumnya [25]. *Openvswitch* mendukung protokol *OpenFlow*, di mana *Openvswitch* sebagai *data plane* dikontrol oleh *controller* sebagai *control plane*. *Data plane* dapat dikontrol untuk melakukan *QoS, tunneling*, dan *filtering rules*. Dukungan pengontrolan jarak jauh membuat *Open vswitch* dapat digunakan untuk melakukan proses migrasi kebijakan jaringan. Karena fleksibilitasnya, data plane pada *Openvswitch* dapat dipartisi secara logik. [25].