BAB 2
DASAR TEORI

2.1 KAJIAN PUSTAKA

Beberapan penelitian terkait perancangan *server* media *streaming* telah dilakukan oleh beberapa peneliti. Penelitian dilakukan menggunakan *server Nginx* dengan mengimplementasikan protokol RTMP sebagai *server* media *streaming* [9], [10]. Perancangan *server* media *streaming* akan di distribusikan oleh HTTP *server* untuk menampilkan hasil layanan *streaming*. Hasil yang diperoleh menunjukkan bahwa kualitas layanan *streaming* yang cukup baik dengan resolusi video *input* 720p, *bitrate* video 800 Kbps, dan audio 128 Kbps pada *bandwidth* 1024 Kbps dengan kualitas gambar yang tidak menurun dan hasil layanan *streaming* tidak mengalami gambar patah-patah. Penggunaan protokol transmisi sangat diperlukan untuk memperoleh kualitas layanan *streaming* dalam jaringan. Terdapat beberapa pilihan protokol transmisi *streaming* yang dapat digunakan, diantaranya HLS, RTMP, dan RTSP [11]. Hasil yang diperoleh pada penelitian [11] dalam kinerja protokol transmisi layanan *streaming*, HLS mempunyai kinerja lebih baik ketika membagi video dalam *fragmen* yang sangat pendek yang berbeda sekitar 4 detik, sedangkan RTSP merupakan protokol yang paling efisien dalam hal konsumsi memori RAM, dan baterai namun mempunyai kelemahan untuk menyesuaikan *bitrate* yang dipengaruhi oleh kecepatan internet. RTMP memiliki hasil yang lebih baik dalam hal stabilitas daripada RTSP, tetapi memiliki kelemahan dari segi kecepatan internet dan penggunaan baterai.

Penerapan metode *load balancing* diharapkan dapat meningkatkan kinerja dari *server* dalam mengalokasikan *workload* dan pendistribusian layanan. Pada penelitian [12] dengan menggunakan *software Haproxy* dan *Nginx* dilakukan pengujian terhadap *server load balancing* untuk *server* E-*learning*. Pengujian dilakukan karena terdapat lonjakan *request* yang disebabkan oleh peningkatan jumlah pengguna yang mengakses E-*Learning* tersebut. Maka, dengan

mengimplementasikan *server load balancing* didapatkan hasil yang menunjukkan bahwa kinerja *Haproxy* lebih baik dibandingkan *Nginx*. Keunggulan kinerja *Haproxy* dapat diketahui berdasarkan parameter *throughput* yaitu 896,48 Kbps lebih besar dari *Nginx* maupun dari parameter *response time* yang didapatkan sebesar 585 ms lebih kecil pada uji koneksi 300/300 sec. Terdapat penelitian terkait penggunaan *load balancing* berdasarkan metode yang digunakan. Pada penelitian [13] dilakukan pengujian untuk membandingkan sistem *load balancing* dengan menerapkan *algoritma round robin* dan algoritma *least connection* pada *web server* dengan mengimplementasikan *haproxy*. Hasil yang diperoleh menunjukkan metode *load balancing* yang digunakan, yaitu *round robin* dan *least connection*. Nilai rata-rata yang didapatkan sama pada parameter *throughput* dari kedua algoritma yang memperoleh 99,5 Kbps berdasarkan pengujian menggunakan *HTTPerf*. Pada pengujian algoritma *least connection* berhasil lebih baik dalam parameter *response time* yaitu sebesar 6,9 ms dan mengunakan *round-robin* memperoleh sebesar 7,2 ms.

Pada penelitian [14] dan [5] *software Openstack* digunakan untuk membangun layanan *server* dengan memanfaatkan teknologi virtualisasi pada layanan *cloud computing*. Hasil yang diperoleh menunjukkan pada layanan *server* video *conference* menggunakan *Openmeetings*, kualitas layanan *throughput* pada *client* mendapatkan sebesar 639,85 Kbps, sedangkan untuk paramater *delay* mendapatkan sebesar 30,22 ms dengan *packet loss* sebesar 0,94% dan parameter *jitter* mencapai sebesar 10,35 ms. Kualitas layanan dari sisi *server* untuk parameter *throughput* mendapatkan sebesar 1,86 Kbps dengan kualitas *delay* mendapatkan sebesar 7,5 ms dengan *packet loss* sebesar 2,9% dan parameter *jitter* mendapatkan sebesar 6,18 ms [14]. Pada penelitian [5] layanan *web server* yang menerapkan metode *load balancing* untuk mengalokasikan *workload server* menggunakan algoritma *least connection* didapatkan hasil dengan nilai rata-rata parameter *throughput* semua variasi sebesar 1,580 Mbps dengan nilai maksimum pada 1000 koneksi *client-server* yaitu 1,859 Mbps. Parameter d*elay* kurang dari 150 ms yang menunjukkan kategori sangat baik berdasarkan standarisasi TIPHON. Presentasi penggunaan CPU tertinggi yaitu pada 5000 koneksi sebesar 62,565%.

2.2 DASAR TEORI

2.2.1 *Streaming*

Streaming adalah metode memutar atau menjalankan *file* media berupa video dan audio menggunakan layanan *server* baik secara langsung atau direkam terlebih dahulu. *Streaming* sangat membantu pengguna dalam memutar *file* audio atau video tanpa mengunduh *file* tersebut terlebih dahulu. *File streaming* diawali dengan melakukan *encoding* terlebih dahulu agar tersedia untuk dikirim melalui jaringan dengan menyesuaikan kecepatan jaringan dan aplikasi yang digunakan untuk memutar *file* tersebut. Video *streaming* merupakan siaran yang dilakukan langsung dan pada waktu yang bersamaan siaran akan di *broadcast* pada banyak orang sesuai dengan kejadian aslinya, melalui media komunikasi jaringan baik yang menggunakan *wireline* maupun yang terhubung *wireless* [15].

Layanan video *streaming* terbagi menjadi 3 jenis berdasarkan bentuknya yaitu:

1. Video *on demand*, layanan video *streaming* ini memperbolehkan pengguna untuk dapat melakukan proses *rewind, fast forward* dan *fast backward* maupun melakukan indeks isi multimedia.
2. *Live streaming*, layanan video *streaming* ini sering ditemui dalam media penyiaran radio dan televisi yang memperbolehkan pengguna untuk menerima siaran secara langsung sebagai pendengar maupun pemirsa.
3. *Real time streaming*, layanan video *streaming* ini memperbolehkan beberapa pengguna dapat saling berkomunikasi dalam waktu yang sama menggunakan layanan audio dan video.

Dalam proses pengiriman informasi, data multimedia mempunyai cara dalam proses transmisi. Terdapat 3 cara yang bisa digunakan dalam jaringan yaitu:

1. *Download mode*, pengguna dapat memutarkan video atau audio setelah semua *file* telah diunduh dari *server*. Dengan menggunakan cara ini, pengguna diharuskan mempunyai keseluruhan *file* secara lengkap yang telah diunduh.
2. *Streaming mode*, pengguna dapat memutarkan video atau audio tanpa melakukan proses unduh terlebih dahulu, namun dapat secara langsung.

Bagian video atau audio yang diterima dari proses transmisi dapat diputar secara langsung.

3. *Progressive download*, pengguna dapat memutarkan video atau audio beberapa detik setelah proses unduh dimulai atau pengguna dapat memutar video atau audio selama dalam proses unduh.

2.2.2 IPTV

Internet Protocol Television (IPTV) adalah kombinasi dari komputer modern, jaringan dan teknologi penyimpanan untuk mengirimkan konten melalui jaringan *Internet Protocol* juga dikenal sebagai IP. *Streaming* video menyumbang 57,69% dari lalu lintas unduhan internet global. Jaringan seluler adalah cara bagi pengguna perangkat seluler untuk mengakses internet secara nirkabel dari mana saja. Selama bertahun-tahun dan evolusi generasi jaringan seluler, telah meningkatkan kecepatan, waktu respons yang telah meningkat, dan menurunkan biaya sehingga menjadi lebih mudah diakses oleh masyarakat umum, yang memungkinkan *streaming* video menjadi salah satu layanan yang paling banyak digunakan di dunia, memungkinkan untuk menikmati konten langsung dari perangkat seluler [11]. IPTV dikembangkan sebagai *prototipe* dan dirancang untuk mendukung pengguna melalui jaringan nirkabel dan/atau seluler, apapun perangkat selulernya. Sistem dapat ditransmisikan baik secara langsung maupun *video-on-demand* (VOD) melalui *browser* seluler (*smart phones, smart televisions*, dan tablet) dan *browser web* (*Windows, Mac,* dan *Unix*).

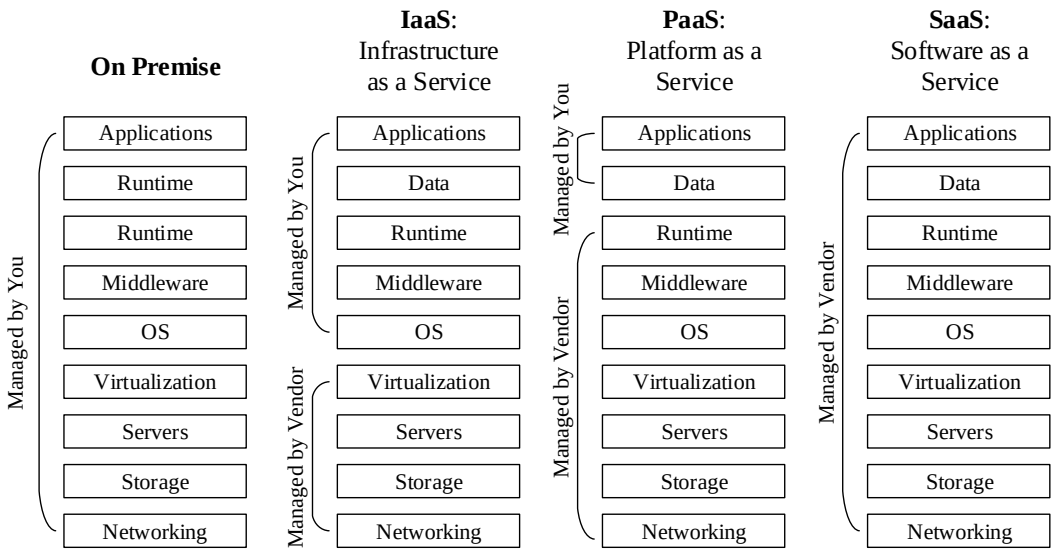
2.2.3 Virtualisasi

Virtualisasi adalah teknologi yang mengizinkan untuk menghilangkan ketergantungan pada perangkat fisik dan komponen lain, sehingga dapat diperlakukan sebagai sesuatu yang virtual. Hal ini dapat menghasilkan perangkat atau aplikasi yang terlihat seperti satu buah komputer fisik, meskipun sebenarnya terdiri dari beberapa komputer yang digabungkan. Berbagai sumber daya yang dapat divirtualisasikan, termasuk perangkat keras komputer, media penyimpanan data, sistem operasi, layanan jaringan, dan daftar ini akan terus berkembang.

Dengan menerapkan teknologi virtualisasi mempunyai keuntungan yaitu dapat meningkatkan efisiensi penggunaan perangkat keras seperti layanan *server* yang dapat dimaksimalkan. Dengan virtualisasi, kinerja perangkat akan menjadi lebih efisien karena sumber daya yang tidak digunakan dapat dikelola dengan lebih baik. Hal ini akan mengurangi ketergantungan pada perangkat keras fisik dari layanan server sistem operasi, dan mengurangi kebutuhan perawatan perangkat keras dan konsumsi daya [16].

2.2.4 *Cloud Computing*

Cloud computing adalah teknologi yang menyediakan kemudahan akses cepat ke kumpulan sumber daya komputasi seperti jaringan, penyimpanan, aplikasi, dan layanan *server* secara bersama-sama. Perkembangan *cloud computing* diawali dari keinginan untuk menyediakan layanan yang fleksibel dengan perangkat virtual sebagai dasar untuk menyediakan berbagai layanan *cloud computing*. *Service* pada layanan *cloud* dapat diartikan sebagai kumpulan fungsi yang disediakan oleh *server* secara virtualisasi yang terdiri dari perangkat keras, perangkat lunak, penyimpanan, kemampuan komputasi, dan infrastruktur. Layanan *cloud computing* biasanya memiliki biaya berlangganan yang ditentukan oleh penyedia layanan, dan hanya memungkinkan diakses pada internet sebagai media komunikasi bagi pelanggan. Dengan layanan *cloud computing* memudahkan dalam mengelola sumber daya secara efektif, dapat memberikan *flexibility* dalam mengembangkan sistem yang kompleks dengan menyesuaikan layanan dan mengelola sistem dengan praktis, dapat menyesuaikan spesifikasi yang diinginkan berdasarkan permintaan pengguna, serta mengurangi penggunaan konsumsi energi dan biaya untuk perawatan perangkat fisik [17].



Gambar 2.1 Tipe Layanan *Cloud Computing* [5]

Pada gambar 2.1 menjelaskan bahwa *service cloud computing* terbagi menjadi 3 jenis, yang terdiri dari:

1. *Infrastructure as a Service* (IaaS)

   IaaS digunakan oleh *administrator*, dengan layanan *application*, *data*, *runtime*, *middleware*, dan OS yang dapat dikelola oleh pengguna, sedangkan layanan *virtualization*, *servera*, *storage*, dan *networking* dibantu kelola oleh *provider*. Adapun penyedia IaaS diantaranya yaitu *Openstack*, *Amazon Web Service*, *Alibaba Cloud*, *Microsoft Azure*, *Google Cloud Platform* dan lain-lain.

2. *Platform as a Service* (PaaS)

   PaaS digunakan oleh *developer* dan *tester*, dengan layanan *application* dan data yang dapat dikelola oleh pengguna, sedangkan layanan *runtime*, *middleware*, OS, *virtualization*, *servera*, *storage*, dan *networking* dibantu kelola oleh *provider*. Adapun penyedia PaaS diantaranya yaitu *Openshift Origin*, *Cloudify*, *Facebook*, *Stackato*, *Github*, dan lain-lain.

3. *Software as a Service* (SaaS)

   SaaS digunakan oleh *end user*, dengan seluruh layanan sudah dikelola oleh *provider*. Adapun penyedia SaaS diantaranya yaitu *Dropbox*, *Google Drive*, *Microsoft One Drive*, dan lain-lain.

Berdasarkan pengembangan, layanan *cloud* terbagi menjadi 4 model, yang terdiri dari:

1. *Public Cloud*

   Model pengembangan bersifat *flexibility*, berbayar, dan peningkatan spesifikasi yang dibutuhkan berdasarkan permintaan pengguna.

2. *Private Cloud*

   Model pengembangan bersifat *full control*, pengguna dapat mengatur sendiri aturan serta fleksibilitas *cloud*-nya sesuai penggunaan dapat berbayar dan *opensource*.

3. *Community Cloud*

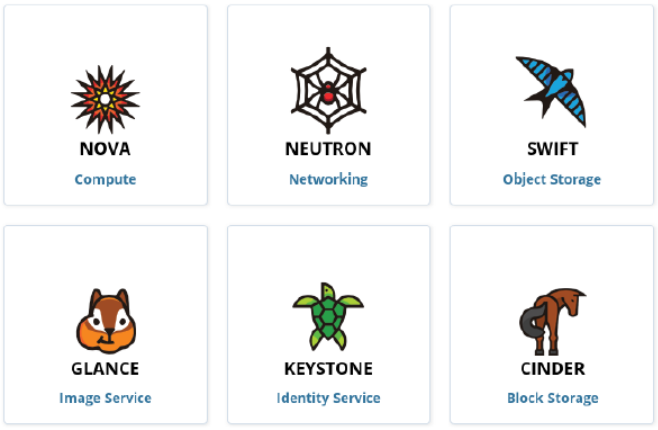
   Model pengembangan berfokus pada berbagi data melalui *cloud* untuk komunitas.

4. *Hybrid Cloud*

   Model pengembangan berupa kombinasi dari *private* dan *public cloud* [5].

2.2.5 *Openstack*

Openstack merupakan perangkat lunak atau *platform* yang bersifat *open-source* (terbuka) dan berfungsi menyediakan layanan *cloud computing*. *Openstack* merupakan *Infrastructure as a Service* (IaaS) yang terdiri dari *server* virtual dan layanan *application, data, runtime, middleware,* dan OS yang disediakan untuk memenuhi kebutuhan pelanggan. *Openstack* merupakan sebuah sistem operasi *cloud* yang menyediakan sumber daya komputasi, penyimpanan, serta jaringan pada pusat data, dan dapat dikelola melalui *dashboard* administrator untuk memantau dan memanajemen sumber daya yang telah dimanfaatkan oleh pengguna [18].



Gambar 2.2 Layanan-layanan *Openstack* [5]

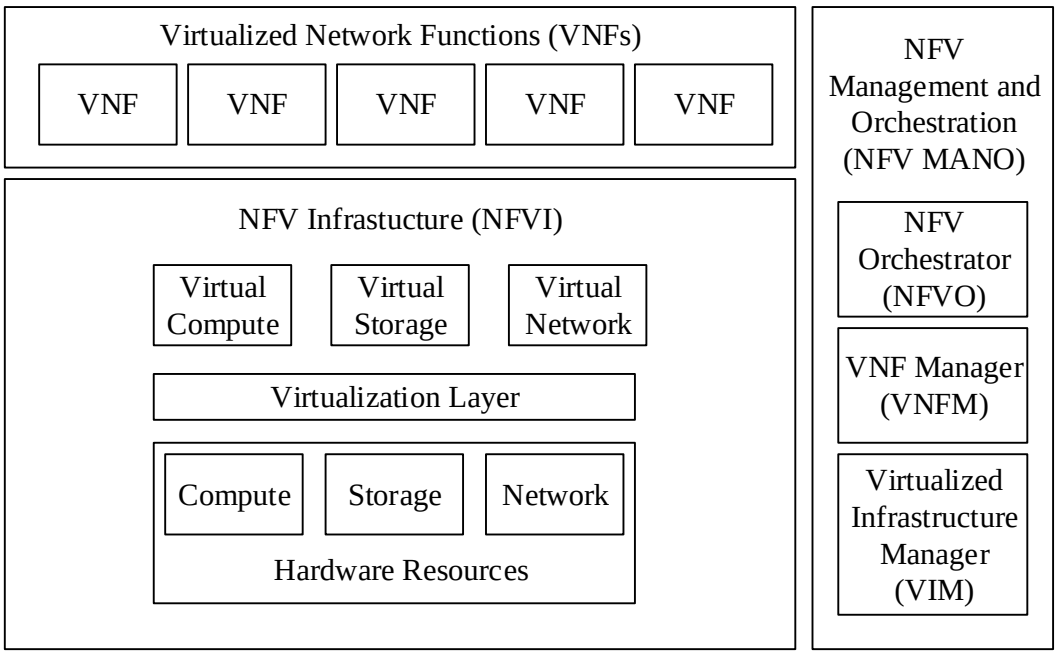
Pada gambar 2.2 menunjukkan bahwa *Openstack* mempunyai layanan-layanan utama yang terdiri dari 6 layanan yaitu:

1. *Nova* (*Compute*), berfungsi memanejemen dan mengatur penggunaan *virtual machine* yang berjalan pada *node hypervisor* serta mengatur alokasi CPU.
2. *Neutron* (*Networking*), berfungsi menyediakan konektifitas jaringan antar *interface* yang ada dalam layanan *Openstack* baik *ingress* maupun *egress*.
3. *Swift* (*Object Storage*), berfungsi mengatur penyimpanan data terdistribusi tanpa titik pusat kontrol yang memberikan skalabilitas yang lebih besar.
4. *Glance* (*Image Service*), berfungsi menyediakan dan menambahkan layanan yang biasa digunakan untuk menyimpan berbagai sumber daya seperti virtual *disk image* yang akan digunakan.
5. *Keystone* (*Identity Service*), berfungsi sebagai layanan terpusat yang mengatur *authentication* dan *authorization* untuk layanan yang ada di *Openstack* serta untuk menejemen *user, project* dan *role*.
6. *Cinder* (*Block Storage*), berfungsi memanejemen penyimpanan yang bersifat tetap untuk *virtual machine*, seperti *harddisk* yang ada pada PC [5].

2.2.6 *Network Function Virtualization* (NFV)

NFV adalah teknologi virtualisasi yang memungkinkan penerapan fungsi jaringan dan tersedia pada perangkat keras standar industri. Hal ini menciptakan

fungsi jaringan yang dapat didistribusikan oleh *provider* layanan dengan hanya berbentuk perangkat lunak yang diinstalasi pada infrastruktur pusat data milik *provider* layanan. Hal ini memungkinkan untuk mengurangi ketergantungan pada perangkat keras jaringan khusus dan meningkatkan fleksibilitas dan efisiensi dalam mengelola jaringan serta mengoptimalkan penggunaan sumber daya jaringan, mengurangi biaya operasional, dan meningkatkan skalabilitas jaringan.



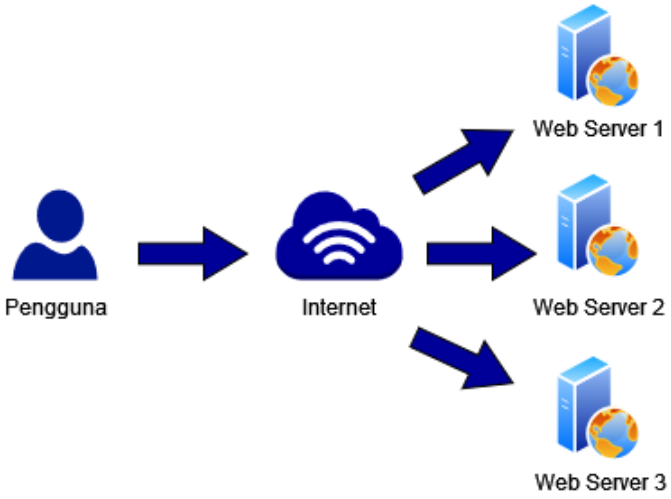
Gambar 2.3 Arsitektur NFV [5]

Pada gambar 2.3 menunjukkan standarisasi sistem dari arsitektur NFV yang diterbitkan oleh ETSI yang terdiri dari 3 komponen utama, yaitu NFVI, NFV MANO, dan VNF. NFV *Infrastructure* (NFVI) berfungsi untuk menyediakan sumber daya virtual yang diperlukan untuk menjalankan *Virtualized Network Functions* (VNF). Komponen dari NFVI termasuk perangkat keras komersial, lapisan *software* yang divirtualisasikan pada perangkat keras tersebut dan komponen akselerator sesuai kebutuhan. NFV *Management and Orchestration* (NFV MANO) bertanggung jawab untuk mengatur dan mengelola *life cycle* dari sumber daya yang digunakan, baik berupa perangkat fisik maupun perangkat lunak yang mendukung pengembangan infrastruktur virtualisasi, serta mengelola *life cycle* dari VNF. NFV MANO berinteraksi dengan jaringan eksternal untuk

memungkinkan integrasi dengan jaringan yang sudah ada. VNF merupakan pengimplementasian dari sebuah fungsi jaringan berbasis perangkat lunak yang dapat dijalankan pada NFVI [5].

2.2.7 *Load Balancing*

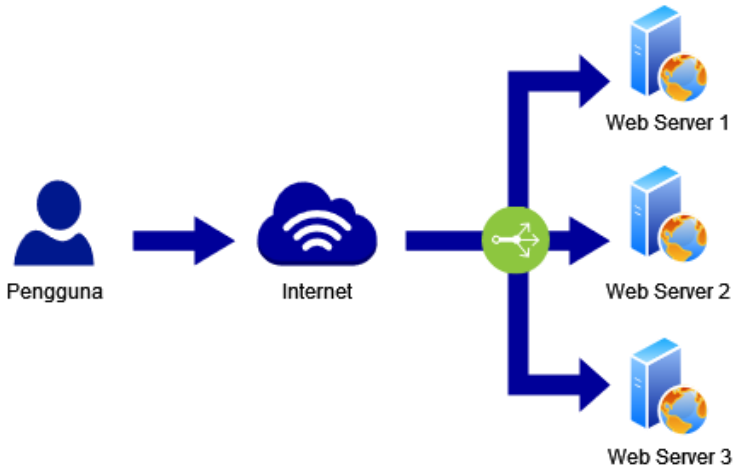
Load balancing adalah metode pembagian *workload* dengan sumber daya yang telah disediakan dalam suatu sistem, sehingga pengalokasian beban sistem pada setiap sumber daya menjadi sama dalam setiap waktu. *Load balancing* bertujuan untuk menyediakan sistem yang memungkinkan mendistribusikan permintaan dari *client* secara seimbang, untuk meminimalkan waktu respons, latensi dan meningkatkan *throughput*. *Load balancing* adalah teknik yang digunakan untuk mendistribusikan sejumlah besar permintaan di beberapa perangkat [19]. Sistem yang tidak mengimpelmentasikan *load balancing* akan menangani setiap permintaan dari *client* hanya pada satu *server* utama saja dan dapat menyebakan antrian seperti gambar 2.4. Jika *server* mengalami gangguan karena terlalu banyak beban, maka akan mengalami penurunan kualitas layanan dari *server* hingga tidak dapat diakses oleh pengguna.



Gambar 2.4 Sistem tanpa *Load Balancing* [5]

Pada gambar 2.5 menunjukkan proses sistem yang mengimplementasikan *load balancing* dan bekerja melayani permintaan dari *client* serta mendistribusikan

ke beberapa *server* dalam satu waktu. Metode *load balancing* hanya dilakukan di antara *server* aktif dari kumpulan server [20]. *Load balancing* berarti membagi lalu lintas *web* dan mendistribusikannya secara merata ke sekelompok *server* yang identik. *Load balancing* sangat penting untuk meng*hosting* aplikasi *web* saat ini. Dengan berbagi pekerjaan, *server* tidak akan terlalu tegang, yang berarti layanan cepat secara konsisten. Hal ini membuat layanan menjadi lebih handal, dengan mengirimkan lalu lintas ke sekelompok *server* dan melindungi *server* dari *downtime* karena jika satu atau bahkan beberapa *server* mengalami kegagalan, layanan masih akan terus berjalan tanpa gangguan selama masih ada *server* lain yang aktif [21].



Gambar 2.5 Sistem Dengan *Load Balancing* [5]

Load balancer meningkatkan kinerja jaringan dengan menggunakan sumber daya yang tersedia dengan benar dan membantu meningkatkan waktu respons dan koneksi per detik. Adapun kelebihan yang didapatkan dalam mengimplementasikan *load balancing* pada sebuah sistem, yaitu :

1) *Cost effectiveness*, *load balancing* bertujuan untuk mendapatkan perbaikan dari keseluruhan sistem dengan biaya yang efektif.
2) *Scalability and flexibility*, dengan menerapkan algoritma untuk mengatur trafik jaringan dapat memberikan layanan yang *scalable* dan fleksibel untuk menangani terjadinya perubahan topologi dengan mudah.

3) *Priority*, dapat memprioritaskan beban kerja pada *server* sesuai dengan algoritma yang digunakan seperti *round robin* atau *least connection* untuk mengatur pembagian tugas yang merata pada setiap *server* [22].

Secara umum, metode *load balancing* mempunyai beberapa algoritma yang dapat digunakan diantaranya:

A. Algoritma *round robin*

Algoritma *Round Robin* adalah algoritma penyeimbangan beban yang paling sederhana dengan cara mendistribusikan setiap permintaan pengguna menuju *server* secara bergiliran. Dalam algoritma *round robin*, proses dibagi di antara semua *processor*. Setiap proses baru ditugaskan ke prosesor baru dalam urutan *round robin*. Algoritma *round robin* diharapkan dapat memberikan kinerja yang tinggi, dengan keandalan yang lebih baik. Namun, ketika pengguna meminta waktu pemrosesan yang tidak sama, algoritma *round robin* akan menjadi kurang efisien karena beberapa *node* dapat menjadi kelebihan beban sementara beberapa *node* tetap menganggur [23].

B. Algoritma *least connection*

Algoritma *least connection* adalah metode penjadwalan yang dinamis, karena memerlukan perhitungan jumlah koneksi yang sedang aktif pada setiap *server* secara *real-time*. Perancangan *server* virtual yang digunakan untuk memanajemen kumpulan *server* dengan kinerja yang sama. Metode *least connection* sangat berguna untuk mendistribusikan permintaan yang terus meningkat dengan lebih efisien [4]. Algoritma *least connection* bekerja dengan mendisrtibusikan permintaan *client-server* berdasarkan *server* yang memiliki jumlah koneksi yang sedang aktif paling sedikit. *Server* yang memiliki jumlah koneksi masuk yang paling banyak akan dialihkan ke *server* lain yang beban kerjanya lebih ringan menggunakan *load balancer*, selain itu algoritma ini mengganggap semua *server* sebagai *backend* yang memiliki kemampuan komputasi yang sama. Algorima ini cocok untuk digunakan pada *cluster* dengan lingkungan yang dinamis dengan *session* yang berubah-ubah [24].

C. Algoritma *weighted round robin*

Algoritma *weighted round robin* merupakan pengembangan dari *round robin*, dengan setiap *server* diberi bobot, yang dihitung berdasarkan kapasitas pemrosesan setiap *node* dapat berupa parameter *power, queue length, job size*, dan sebagainya. Berdasarkan bobotnya, *server* diurutkan dari yang paling tinggi ke yang paling rendah. *Server* dengan bobot yang lebih besar diberi lebih banyak permintaan dibandingkan dengan *server* dengan bobot yang lebih kecil. Pendekatan ini menghasilkan kinerja dan *throughput* yang lebih baik [25].

D. Algoritma *weighted least connection*

Algoritma *weighted least connection* merupakan pengembangan dari algoritma *least connection*, dengan menetapkan bobot kinerja untuk setiap *server* nyata. *Server* dengan nilai bobot yang lebih tinggi akan menerima persentase yang lebih besar dari koneksi yang lebih besar pada satu waktu. Pada algoritma ini, sebuah jaringan baru dari koneksi baru diberikan ke *server* yang memiliki rasio minimum dari jumlah koneksi aktif saat ini dengan bobotnya [23].

2.2.8 *Nginx*

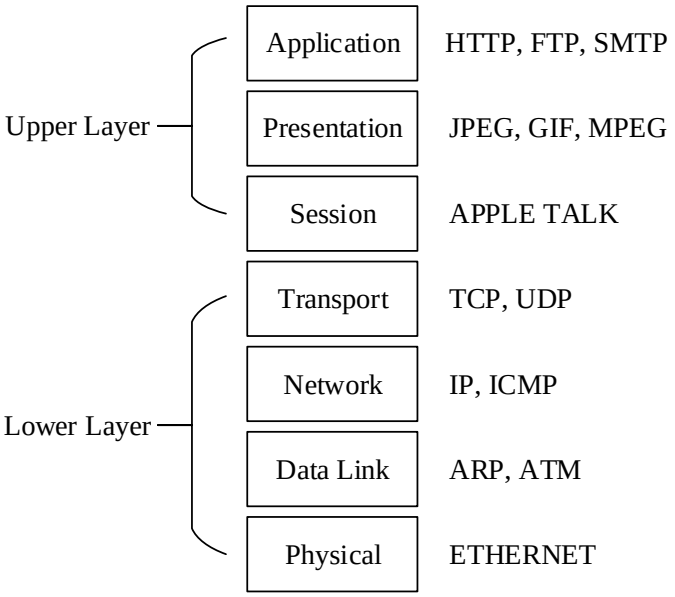
Nginx adalah perangkat lunak *open-source* yang memiliki performansi tinggi sebagai HTTP dan *reverse proxy server*. Selain itu, *Nginx* mempunyai fungsi lain yaitu dapat menyediakan *streaming* media *server*. *Nginx* dikenal sebagai solusi yang cepat dan handal untuk menangani trafik *web* yang tinggi dan dapat digunakan untuk meningkatkan kinerja dan skalabilitas aplikasi *web*. *Nginx* juga dapat digunakan sebagai *web server* utama atau sebagai *web server* yang digunakan sebagai *front-end* dari layanan aplikasi berbasis *web*. *Nginx* mempunyai kemampuan dalam menyediakan konten statis dengan penggunaan sumber daya sistem yang efisien. *Nginx* dapat dijalankan dan tersedia untuk berbagai *platform* seperti *Microsoft Windows*, distro *Linux*, *Unix*, varian dari BSD, dan *MacOS* [26].

2.2.9 *Web Server*

Web Server adalah perangkat lunak yang menggunakan protokol aplikasi HTTP atau HTTPS sebagai media dalam mengakses atau menyediakan *file-file* dalam halaman *web* melalui *browser*. Penggunaan *web server* sebagai tempat untuk menampilkan antarmuka layanan yang berupa penyimpanan atau *database*, serta digunakan untuk berbagai layanan aplikasi [27]. Dalam pengimplementasian *web server* dapat dibagi menjadi 2 tipe yaitu *web* statis dan *web* dinamis. *Web* statis merupakan sebuah *web server* yang tidak mempunyai *database* dan keterbatasan interaksi terhadap *client* serta biasanya dibangun menggunakan HTML, CSS dan *javascript*. Sedangkan *web* dinamis merupakan sebuah *web server* yang mempunyai *database* dan antarmuka yang dimuat dapat berinteraksi dengan *client* serta sangat kompleks untuk membangunnya seperti menggunakan *javascript*, PHP, MySQL dan lain-lain. *Web* statis memiliki kelebihan yaitu dapat mudah dimuat lebih cepat oleh *browser*. Diantara *software* yang paling populer digunakan untuk *web server* yaitu *Nginx* dan *Apache*. Dengan menggunakan *software tersebut* sudah dapat menjadi dasar membangun *web server* karena dengan merubah struktur HTML dan tampilan CSS, *web server* sudah dapat berjalan dan dapat diakses menggunakan protokol HTTP.

2.2.10 OSI *Layer*

Lapisan OSI (*Open System Interconnection*) adalah salah satu arsitektur jaringan yang umum digunakan untuk menerangkan cara kerja jaringan komputer dapat berupa pengertian maupun protokol secara logis. Lapisan OSI terbagi menjadi beberapa fungsi jaringan yang terdiri dari 7 lapisan/*layer* seperti gambar 2.6. Organisasi yang merilis OSI *layer* adalah *International Organization for Standardization* (ISO) dan diperkenalkan pada tahun 1984.



Gambar 2.6 OSI *Layer* [28]

Adapun susunan 7 bagian dari lapisan OSI diantaranya, yaitu:

1. *Application Layer* berfungsi menyediakan *interface* antara aplikasi yang digunakan untuk berkomunikasi dengan jaringan yang digunakan dalam mentransmisikan paket/pesan. Protokol lapisan aplikasi digunakan untuk bertukar data antara program yang berjalan pada *host* sumber dan tujuan.
2. *Presentation Layer* berfungsi untuk menyajikan data ke lapisan aplikasi. Lapisan presentasi seperti penerjemah jaringan.
3. *Session Layer* berfungsi untuk menetapkan dan mengakhiri sesi (*sesion*) antara dua atau beberapa *host* yang saling berkomunikasi.
4. *Transport Layer* berfungsi memastikan pesan yang dikirim bebas dari kesalahan dan pada layer ini digunakan untuk mengatur protokol yang akan digunakan dalam proses transmisi paket/pesan.
5. *Network Layer* berfungsi menyediakan layanan untuk bertukar satu bagian data melalui jaringan antara perangkat-perangkat akhir.
6. *Data Link Layer* berfungsi menyediakan sarana untuk bertukar data melalui media lokal umum.
7. *Physical Layer* berfungsi sebagai media transmisi jaringan ke media fisik dan membawa sinyal ke lapisan yang lebih tinggi [28].

2.2.11 *Transmission Control Protocol* (TCP)

Pada lapisan *transport* terdapat dua protokol utama yang umum digunakan yaitu *Transmission Control Protocol* (TCP) dan *User Datagram Protocol* (UDP). TCP merupakan protokol yang populer digunakan untuk layanan yang ada di internet saat ini. Namun, pada protokol ketika jaringan mengalami kepadatan atau sibuk akan menyebabkan koneksi *timeout* dan retransmisi karena sifatnya yang mengutamakan koneksi. UDP merupakan protokol yang digunakan untuk mendapatkan kecepatan pengiriman data tanpa memperhatikan kontrol kongesti dan koreksi kesalahan dalam jaringan. Meskipun kecepatan pengiriman data tidak dapat dikendalikan, protokol UDP akan sangat berpengaruh terhadap *bandwidth* yang tersedia dalam jaringan, karena protokol ini mengambil seluruh *bandwidth* untuk mendapatkan kecepatan pengiriman data yang maksimum. Protokol UDP pada dasarnya hanya mengandung *header* IP yang singkat. Protokol UDP tidak melakukan proses kontrol aliran data, kontrol kesalahan atau pengiriman ulang terhadap kesalahan, sehingga hanya menyediakan antarmuka ke protokol IP tanpa memastikan informasi dalam keadaan utuh. Protokol TCP memiliki kemampuan dalam memastikan setiap data yang dikirimkan untuk sampai dengan keadaan utuh kepada penerima [29].

2.2.12 Protokol Transmisi Media

HTTP Live Streaming (HLS) adalah protokol komunikasi *bitrate* adaptif berbasis HTTP yang dikembangkan oleh *Apple Inc*. Dukungan protokol tersebar luas di pemutar media, *browser web*, perangkat seluler, dan *server media streaming*. HLS membagi aliran keseluruhan menjadi urutan *file* kecil berbasis HTTP unduhan, setiap unduhan mengunggah sebagian kecil dari aliran transportasi global yang berpotensi tidak dibatasi. Daftar aliran yang tersedia, dikodekan pada kecepatan bit yang berbeda, dikirim ke *client* menggunakan daftar putar M3U yang diperluas. Konten tersedia untuk pengguna pada berbagai *bitrate* yang berbeda, sehingga pengguna dapat memilih *file* berbasis HTTP dengan *bitrate* tertinggi yang dapat diunduh tepat waktu untuk pemutaran tanpa menyebabkan penundaan.

Real Time Streaming Protocol (RTSP) menggunakan *server* untuk melacak status *client* dan menyediakan *streaming. Real Time Message Protocol* (RTMP) dikembangkan untuk transmisi audio, video, dan data berkinerja tinggi. RTMP saat ini tersedia dan bersifat terbuka untuk membuat produk dan teknologi yang memungkinkan pengiriman video, audio dan data dalam format AMF, FLV, SWF, dan F4V terbuka dengan didukung oleh *Adobe Flash Player*. Sementara HLS dan RTMP beroperasi melalui aliran transportasi yang handal, seperti TCP sedangkan RTSP biasanya beroperasi melalui UDP [11].

2.2.13 *Multicast*

Transmisi *multicast* adalah teknologi yang dapat menghemat *bandwidth* dengan mentransmisikan *stream* tunggal ke beberapa pengguna atau *host* secara bersamaan. Dalam proses transmisi, aplikasi yang memerlukan *bandwidth* besar seperti *streaming*, metode *multicast* dapat menjadi solusi untuk mengatasi permasalahan *bandwidth* terbatas dan ketersediaan jaringan terhadap paket *stream* yang cenderung besar dan kontinyu. Keunggulan utama dari komunikasi *multicast* dibandingkan dengan komunikasi lain adalah kemampuan untuk menghindari pengiriman data yang sama berulang-ulang pada *link* yang sama [30].

2.2.14 *Quality Of Service* (QOS)

Quality of Service atau QoS adalah metode penilaian yang berkaitan dengan kualitas layanan berupa seberapa baik suatu jaringan. QoS merupakan cara dalam menentukan karakteristik dan kategori berdasarkan kualitas suatu layanan dari sisi *client* maupun *server*. TIPHON (*Telecommunications and Internet Protocol Harmonization Over Network*) merupakan standarisasi penilaian untuk parameter QoS, yang diterbitkan oleh badan standar ETSI (*European Telecommunications Standards Insitute*) [31].

A. *Throughput*

Throughput menunjukkan kapasitas *bandwidth* yang sebenarnya diukur pada waktu tertentu dan dalam kondisi jaringan tertentu saat terjadi proses transfer

file dengan ukuran tertentu. *Throughput* sistem adalah jumlah kecepatan data yang diterima oleh semua terminal di dalam jaringan [32]. Berdasarkan persamaan, *throughput* diukur sebagai jumlah total paket yang sukses tiba pada tujuan dalam interval waktu tertentu dibagi dengan durasi interval waktu tersebut [31]. Nilai *throughput* dapat dihitung menggunakan persamaan 2.1.

$$Throughput = \frac{\text{Jumlah paket data yang dikirim (bit)}}{\text{Waktu pengiriman paket } (second)} \text{ bps} \qquad \textbf{(2.1)}$$

Penggolongan karakteristik *throughput* dibedakan berdasarkan total bit data yang dikirim dalam waktu perdetik, seperti yang ditunjukkan dalam tabel 2.1.

Tabel 2.1 Kategori *Throughput* [33]

Kategori	***Throughput***
Sangat baik	>2,1 Mbps
Baik	1200 Kbps – 2,1 Mbps
Sedang	700 – 1200 Kbps
Buruk	338 – 700 Kbps
Sangat Buruk	0 – 338 Kbps

Beberapa faktor yang mempengaruhi kualitas layanan *throughput* dalam jaringan meliputi:

1. Kecepatan *processor*: penggunaan *processor* yang lebih cepat akan dapat memproses data lebih cepat dan meningkatkan *throughput*.
2. Kecepatan memori: akses ke memori yang lebih cepat akan memungkinkan *processor* untuk bekerja lebih efisien.
3. Jumlah koneksi jaringan: sistem dengan jumlah koneksi jaringan yang lebih banyak dapat menangani lebih banyak data secara bersamaan.
4. Pembagian tugas: pembagian tugas yang efisien dapat memungkinkan sistem untuk menangani lebih banyak data secara bersamaan.
5. Konfigurasi sistem : Konfigurasi sistem yang tepat dapat meningkatkan *throughput* dengan memastikan bahwa sumber daya yang dibutuhkan tersedia pada saat diperlukan [34].

B. *Delay*

Delay merupakan waktu yang dibutuhkan data untuk menempuh perjalanan dari sumber ke tujuan. *Delay* dapat dipengaruhi oleh jarak, media fisik, kongesti jaringan, atau waktu proses/antrian yang lama [31]. Nilai *delay* dapat dihitung menggunakan persamaan 2.2.

$$\textit{Delay} \text{ rata-rata } = \frac{\text{Total } delay}{\text{Total paket yang diterima}} \text{ ms} \qquad \textbf{(2.2)}$$

Penggolongan karakteristik *delay* dibedakan berdasarkan tingkat kepuasan pengguna, seperti yang ditunjukkan dalam tabel 2.2.

Tabel 2.2 Kategori *Delay* [33]

Kategori	***Delay***
Baik	0 – 150 ms
Sedang	150 – 400 ms
Buruk	>400 ms

Beberapa faktor yang mempengaruhi kualitas layanan *delay* dalam jaringan meliputi:

1. Jumlah *hops*: semakin banyak *hops* yang dilalui paket data, semakin besar *delay* yang terjadi.
2. Kecepatan jaringan: kecepatan jaringan yang lebih rendah akan menyebabkan *delay* yang lebih besar.
3. Beban jaringan: jika trafik jaringan mengalami peningkatan, maka *delay* akan lebih besar karena paket data harus menunggu untuk diteruskan.
4. *Routing* jaringan : konfigurasi jaringan yang salah dapat menyebabkan *delay* yang lebih besar karena paket data harus melewati rute yang tidak efisien.
5. Jumlah pengguna: jumlah pengguna yang banyak dapat menyebabkan *delay* yang lebih besar karena paket data harus menunggu untuk diteruskan [35].

C. *Packet Loss*

Packet loss merupakan parameter yang menggambarkan kondisi dimana ada jumlah total paket yang hilang. Salah satu penyebab *packet loss* adalah ketika antrian melebihi kapasitas *buffer* pada setiap *node* dan menyebabkan *timeout* [31]. Nilai packet loss dapat dihitung menggunakan persamaan 2.3.

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

Penggolongan kategori *packet loss* dibedakan berdasarkan persentase jumlah paket data yang hilang, seperti yang ditunjukkan dalam tabel 2.3.

Tabel 2.3 Kategori *Packet Loss* [33]

Kategori	***Packet Loss* (%)**
Sangat baik	0 – 2
Baik	3 – 14
Sedang	15 – 24
Buruk	>25

Beberapa faktor yang mempengaruhi kualitas layanan *packet loss* dalam jaringan meliputi:

1. *Congestion*: Jika jumlah data yang dikirim melebihi kapasitas jaringan, maka akan terjadi *congestions* dan beberapa *packet* akan hilang.
2. *Routing issues*: Jika ada masalah dengan *routing*, *packet* dapat hilang saat melewati jalur yang salah atau tidak sesuai.
3. *Overloaded router/switch*: Jika *router* atau *switch* yang digunakan terlalu sibuk atau *overload*, maka beberapa *packet* dapat hilang.
4. *Security*: *firewall* yang terlalu ketat atau pengaturan keamanan yang salah dapat menyebabkan *packet loss*.

E. CPU *Usage*

Penggunaan CPU atau CPU *usage* adalah persentase dari total kapasitas CPU yang digunakan saat suatu proses sedang berjalan di *server*. Pengujian CPU *usage* mempunyai tujuan untuk mengevaluasi pembagian beban pada setiap *web server* oleh *load balancer*, serta mengetahui performa baik dalam menangani permintaan masuk maupun keluar dari *web server* [5]. Penggunaan CPU sangat diperlukan untuk mengetahui kinerja dari sistem yang bekerja. Ketika CPU yang bekerja secara berlebihan atau mengalami *overload* maka akan berpengaruh pada layanan yang dihasilkan akan menjadi lambat untuk diakses. Berdasarkan CPU *usage* pada sistem *load balancing* menggunakan *Haproxy* [36], ketika *core* mengalami jenuh maka sistem dapat dibagi menjadi 3 tipikal yaitu:

- 95% sistem dan 5% *user* untuk koneksi TCP panjang atau objek HTTP besar.
- 85% sistem dan 15% *user* untuk koneksi TCP pendek atau objek HTTP kecil dalam mode tertutup.
- 70% sistem dan 30% *user* untuk objek HTTP kecil dalam mode tetap hidup.

F. *Response Time*

Response Time merupakan periode waktu yang diperlukan untuk menyelesaikan setiap permintaan dan memberikan tanggapan dari komunikasi *server-client* [16]. Pengujian *response time* bertujuan untuk mengukur dan memantau seberapa cepat *server* dapat mengatasi jumlah koneksi berupa *request* yang datang dari komunikasi *client-server*. Suatu *server* akan bekerja mengirimkan layanan berdasarkan *request* yang dikirimkan *client*. *Response time* diperlukan untuk mengetahui nilai waktu yang dibutuhkan *server* dalam melayani *request* dan koneksi yang digunakan untuk mengetahui kinerja dari *server*.