BAB II

TEORI PENUNJANG

Di dalam bab ini, penulis mengambil beberapa tinjauan pustaka yang dijadikan sebagai landasan pengerjaan aplikasi *Link Aggregation* dengan *Link Sharing* bermetode *Weighted Round Robin* pada *Software Defined Networking (SDN)* dengan menggunakan *Ryu Controller*.

2.1 Software Defined Networking (SDN)

Software Defined Networking (SDN) merupakan sebuah pendekatan arsitektur jaringan komputer yang memisahkan *control-plane* dari sebuah perangkat jaringan komputer (*Switch* atau *Router*) dengan *data-plane* perangkat jaringan komputer tersebut [4]. Pemisahan *data-plane* dan *control-plane* ini memungkinkan untuk memprogram perangkat tersebut sesuai dengan yang diinginkan secara terpusat (*SDN Controller*), sehingga hal ini memungkinkan untuk mengontrol, memonitor, dan mengatur sebuah jaringan komputer dari sebuah titik (*node*) terpusat tersebut.

2.1.1 Konsep Dasar Software Defined Networking

Software-Defined Networking (SDN) adalah sebuah konsep pendekatan jaringan komputer dimana sistem pengkontrol dari arus data (*control-plane*) secara fisik dipisahkan dari perangkat kerasnya (*data-plane*).

Umumnya, sistem pembuat keputusan kemana arus data akan dikirimkan dibikin menyatu dengan perangkat kerasnya. ***SDN*** memungkinkan penggunanya untuk menulis aplikasi untuk mengelola layanan jaringan termasuk *load-balancing, routing, access control, multicast,* dan tugas-tugas rekayasa lalu-lintas *(traffic)* lainnya.

SDN memerlukan beberapa metode agar *control-plane* dapat berkomunikasi dengan *data-plane*. Salah satu mekanisme tersebut adalah *OpenFlow* yang sering disalahpahami setara dengan *SDN*. *OpenFlow* adalah sebuah protokol yang memungkinkan pengaturan penjaluran dan pengiriman paket data ketika melalui sebuah *Switch*. Dalam sebuah jaringan konvensional, setiap *Switch* hanya berfungsi meneruskan paket data yang lewat ke *port* yang sesuai tanpa dapat membedakan tipe protokol data yang dikirimkan misalnya *elastic* atau *inelastic traffic*.

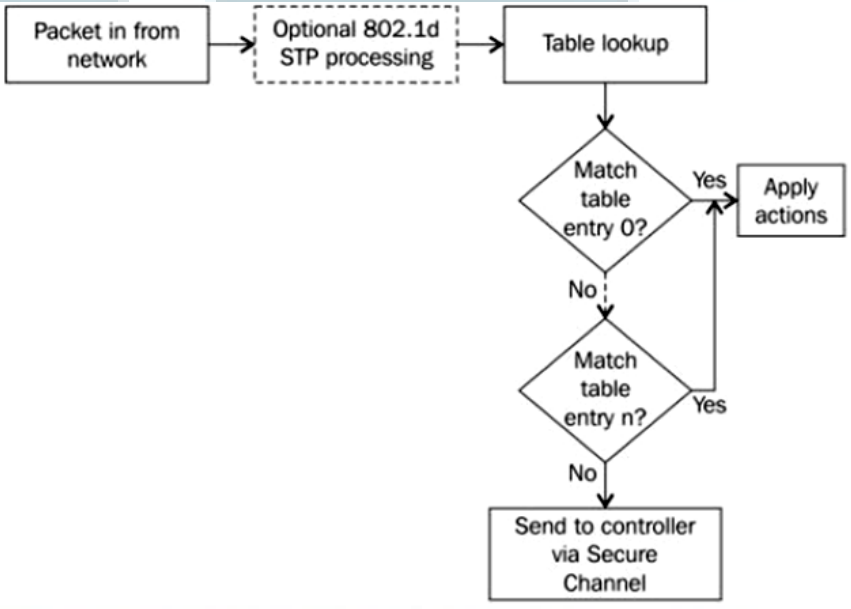
Melalui *OpenFlow,* kita tidak hanya dapat melakukan *flow forwarding* berbasis *network layer* tetapi juga dapat dilakukan pengaturan pergerakan paket data secara terpusat mulai dari *layer* 2 sampai *layer* 7 *forwarding* (*flow granuality*), sehingga aliran paket data di jaringan dapat diprogram secara independen [5]. Hal ini dapat dilakukan dengan membuat algoritma dan *forwarding rules*-nya pada *controller* kemudian aturan tersebut didistribusikan ke *switch* yang ada di jaringan. Terdapat beberapa *OpenFlow Controller* yang dapat digunakan seperti *NOX (C base), Ryu (phyton base),* dan *Floodlight (java base).*

Berikut adalah proses aliran paket data pada *Switch Openflow* [6]:

a. Ketika sebuah paket data tiba di *switch OpenFlow*, bagian *header* paket diperiksa berdasarkan entri *flow table.*

b. Jika entri yang sesuai ditemukan, *Switch* kemudian menerapkan instruksi-instruksi terkait berdasarkan aliran paket data yang sesuai juga.

c. Jika tidak ada yang sesuai dengan prosedur *flow table*, maka aliran paket data akan diarahkan ke entri *table-miss*. Entri *table-miss* adalah entri yang diperlukan yang menentukan set instruksi yang akan diterapkan terhadap paket data yang masuk ketika tidak ada yang cocok atau sesuai dengan prosedur *flow table*. Instruksi mencakup: Menghapus (*dropping*) paket; Mengirim paket pada semua *interface*; dan Meneruskan paket ke *controller.*

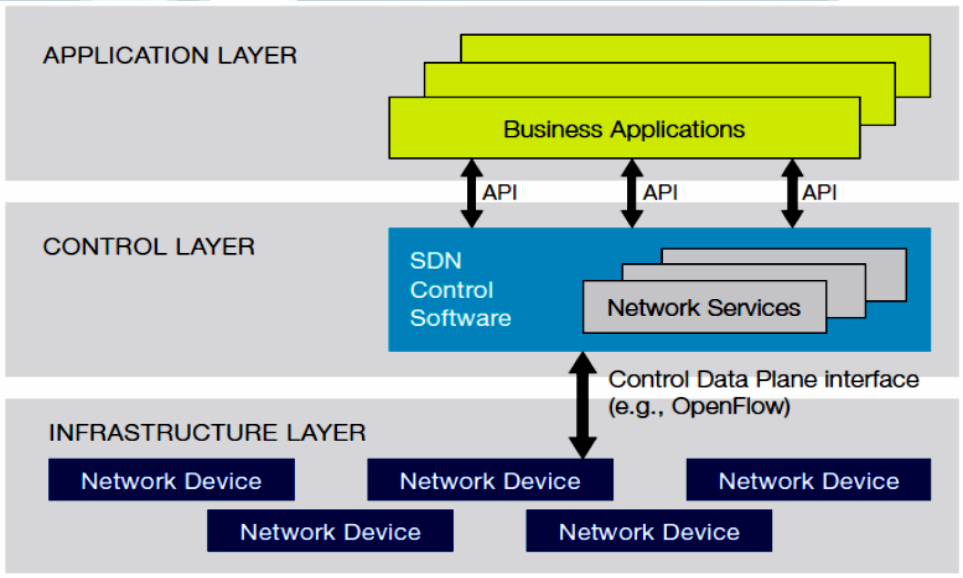


Gambar 2.1 *Flowchart* Arus Paket dalam *Switch OpenFlow* [2]

2.1.2 Arsitektur Software Defined Networking (SDN)

Dalam arsitektur *Software Defined Networking (SDN),* setiap *layer* dapat bekerja secara independen dan berkomunikasi melalui antarmuka jaringan untuk memberikan fungsi berlapis dari perangkat fisik yang berbeda [2]. Aspek arsitektur ini memungkinkan administrator jaringan untuk mengatasi beberapa tantangan dalam dunia jaringan komputer.

Gambaran logis dari arsitektur *Software Defined Networking (SDN)* dapat dilihat pada Gambar 2.2. ini. Terdapat tiga *layer* pada arsitektur *Software Defined Networking (SDN)* [2].



Gambar 2.2 Arsitektur *Software Defined Networking (SDN)* [2]

Keterangan Gambar 2.2. (dari *Layer* paling bawah ke atas):

a. Layer Infrastruktur (*Infrastructure Layer*)

Layer infrastruktur terdiri dari elemen-elemen jaringan dan perangkat keras yang menjalankan fungsi *packet switching* dan *forwarding.*

d. Layer Kontrol (*Control Layer*)

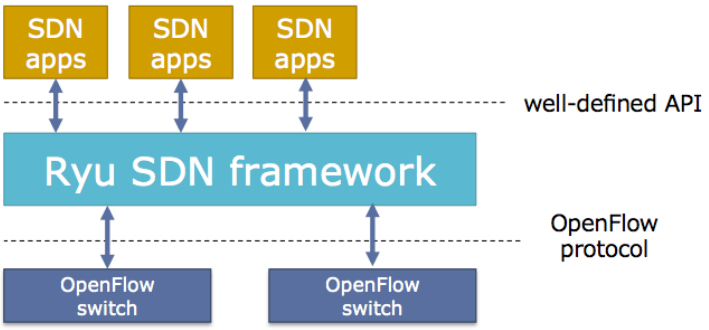
Layer Kontrol menyediakan fungsionalitas kontrol secara padu yang mengawasi perilaku jaringan *forwarding* melalui *open interface*.

e. Layer Aplikasi (*Application Layer*)

Layer Aplikasi berfungsi untuk menyediakan *interface* dalam pembuatan program aplikasi yang kemudian akan mengatur dan mengoptimalkan jaringan secara baik dan fleksibel.

2.2 Ryu Controller

Dalam anologi Sistem Operasi, *OpenFlow* bertindak sebagai sistem operasi dan harus mengimplementasikan dua *interface*, yaitu: 1)*southbound interface* yang memungkinkan *Switch OpenFlow* berkomunikasi dengan *controller (POX, OpenDaylight, FloodLight, Ryu,* dan sebagainya); dan 2)*northbound interface* yang menyajikan *Application Programming Interface (API)* yang dapat diprogram untuk mengontrol jaringan dan memanajemen aplikasi (*Pyretic, Frenetic*) [7].



Gambar 2.3 *Ryu Controller Layer* dalam Arsitektur *SDN* [7]

Ryu Controller adalah *open, controller Software Defined Networking (SDN)* yang dirancang untuk meningkatkan kecepatan jaringan dengan membuatnya mudah untuk mengelola dan beradaptasi dengan trafik *(traffic)* jaringan. Secara umum, *SDN Controller* adalah otak dari *SDN*, yang mengkomunikasikan informasi ke bawah *Switch* dan *Router* dengan *southbound API* dan ke atas untuk aplikasi dan logika bisnis dengan *northbound API* [7].

Source code dari *Ryu Controller* di-*hosting* di GitHub dan dikembangkan oleh komunitas terbuka *Ryu*. *OpenStack*, yang berkolaborasi bersama untuk membangun *cloud operating system* yang bisa mengendalikan komputasi, penyimpanan, dan jaringan dari sebuah organisasi, mendukung pengembangan *Ryu* sebagai *controller SDN* [7].

Gambar 2.4 Logo *Ryu Controller* [8]

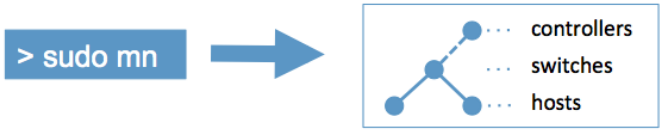
Ditulis sepenuhnya dengan bahasa *Python*, *source code* tersedia dengan lisensi *Apache 2.0* dan terbuka untuk siapa saja yang ingin mengembangkan. *Ryu Controller* bisa menggunakan *OpenFlow* untuk berinteraksi dengan *Switch* dan *Router* untuk memodifikasi bagaimana cara menangani trafik jaringan. *Ryu* telah diuji dan bersertifikat untuk bekerja dengan berbagai macam jenis *OpenFlow Switch*, termasuk *Open vSwitch, Hewlett Packard, IBM* dan *NEC* [7].

2.3 Mininet

Mininet adalah *network emulator* atau lebih tepatnya disebut sebagai *network emulation orchestration system* yang terbuka dan mendukung protokol *OpenFlow* untuk arsitektur *SDN*. *Software* ini merupakan *emulator* yang terpopuler di kalangan pengembang *SDN*. *Mininet* bekerja dengan cara membuat *virtual hosts, switches, controllers,* dan *links* pada *single Linux kernel*. *Host mininet* bertindak layaknya mesin sesungguhnya, bisa diakses menggunakan *SSH* dan mengirim paket apa saja melalui *Ethernet Interface* yang bisa diberi *bandwidth* dan *delay*. Paket akan diproses layaknya di *ethernet switch, router,* atau *middlebox* sesungguhnya [9].

Kelebihan lainnya, *Mininet* bisa membuat *custom topologies* dari *single switch* hingga layaknya seperti *backbone*. *Mininet* mampu menjalankan program sesungguhnya yang berjalan pada *Linux,* yang bisa dijalankan dari *web server* hingga TCP *window monitoring tools* seperti *Wireshark*. *Mininet* bisa juga dijalankan di *laptop* atau *server* atau dalam *Virtual Machine (VM)* dengan *native Linux* [9].

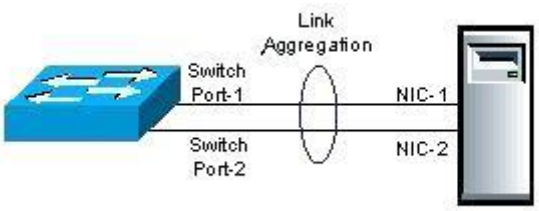
Bahasa yang digunakan *Mininet* sebagian besar *Python*. *Mininet* adalah *open source project*, jadi semua orang bisa melihat dan mengembangkan *source code* yang tersedia di situs *https://github.com/mininet*. Berjalan berdasarkan *standard Linux/Unix network application* seperti *Linux kernel* dan *network stack* yang membuat kode yang dikembangkan dan diuji di Mininet untuk *OpenFlow controller, modified switch,* atau *host* bisa dipindahkan ke sistem sungguhan dengan sedikit perubahan untuk pengujian di dunia nyata, evaluasi performa, dan *deployment* [9].



Gambar 2.5 *Command* Utama *Mininet* [9]

2.4 Link Aggregation

Link Aggregation (LAg) adalah bahasa standar teknologi jaringan komputer yang menggabungkan beberapa *link*/*trunk*/*cable*/*port* fisik secara paralel untuk mendapatkan kapasitas *Bandwidth* yang lebih besar dan untuk memberikan *redundancy* dengan menggunakan teknologi *Ethernet.* Beberapa *vendor* memiliki standar masing-masing untuk penerapan *LAg*, namun standar internasional yang umum digunakan untuk *LAg* adalah *IEEE802.3ad*. Istilah lain *LAg* ini adalah *Link Aggregation Control Protocol (LACP).*



Gambar 2.6 Contoh Penerapan *Link Aggregation* pada *Server* [11]

LAg bisa diterapkan pada hubungan antara Komputer dengan *Switch*, Komputer dengan *Router*, *Switch* dengan *Switch*, *Router* dengan *Router*, dan beberapa konfigurasi yang lainnya. Protokol untuk *LAg* lain selain *LACP* adalah *PAgP*, tetapi itu *proprietary* milik *vendor cisco* dan tidak bisa dipergunakan secara umum. *LACP* bisa dikonfigurasi dengan dua cara [12], sebagai berikut:

a. *Active mode* adalah ketika perangkat secara terus menerus mengirim *LACP data unit* ketika *port* aktif.

b. *Passive mode* adalah ketika perangkat yang tidak mengirim LACP *data unit* melainkan hanya membalas setiap *LACP data unit* yang diterima, tetapi tidak berinisisasi untuk mengawali negosiasi *LACP*.

Penggabungan beberapa *link* menggunakan *LAg* harus dari *link* yang memiliki kapasitas yang sama, seperti sesama 10Mb/s atau sesama 100Mb/s. Kapasitas yang lain yang memungkinkan untuk digabungkan adalah 1Gb/s dan 10Gb/s. Kombinasi dari berbagai kapasitas tidak dimungkinkan. Peningkatan kapasitas dalam *LAg* sesuai dengan jumlah *link* yang digunakan terhadap kapasitas individu *link*. Misal, penggabungan tiga *link* 10Mb/s akan mendapatkan 30Mb/s. Begitu juga dengan penggabungan lima *link* 100Mb/s akan mendapatkan kapasitas 500Mb/s. Proteksi *link* menggunakan *LAg* tidak dimaksudkan untuk menambah kapasitas tetapi adalah untuk *high availability*.

Pembuatan *LAg* bisa dimulai dari dua atau tiga *link*. Jumlah maksimum *link* yang bisa digabungkan untuk protokol *LACP* yaitu delapan buah. Umumnya, setiap *vendor* memiliki kemampuan minimal empat *link* untuk digabungkan. *Interface Aggregation* akan terlihat sebagai *single interface*. Utilisasi *interface aggregation* adalah jumlah dari paket-paket yang dikirimkan pada tiap-tiap *link*. Utilisasi masing-masing *link* adalah paket-paket yang dikirimkan pada masing-masing *link* tersebut. Umumnya paket akan melewati *link* tergantung pada *hashing*, dan *hashing* yang umum digunakan adalah per-*destination IP*.

2.5 Weighted Round Robin

Penjadwalan *Weighted Round Robin* merupakan evolusi dari penjadwalan *Round Robin*. Setiap *Link* dapat diberi bobot berupa *integer* yang mengindikasikan kapasitas performanya. *Link* yang mempunyai bobot lebih tinggi akan mendapatkan jatah terlebih dahulu dari pada *Link* yang mempunyai bobot lebih rendah dan *Link* yang mempunyai bobot lebih besar akan mendapatkan mendapatkan jatah lebih banyak dibandingkan dengan *Link* yang mempunyai bobot lebih kecil [13].

Terdapat dua teknik penjadwalan WRR yaitu:

A. *Busty Service*: Teknik penjadwalan yang mengirim paket data dengan tidak merata. Contoh: AAAABBBCC.
B. *Smooth Service*: Teknik penjadwalan yang mengirim paket data dengan merata. Contoh: AABABCABC.

Satu siklus WRR dapat diketahui dari total bobot prioritasnya. Contoh, terdapat tiga *Link* yaitu A, B, dan C yang masing-masing mempunyai bobot 4, 3, dan 2. Maka, 1 siklus WRR adalah 9 dari penjumlahan 4 + 3 + 2 = 9. Proses distribusi pembagian jatah dimulai dari prioritas tertinggi ke terendah dengan proses giliran *Round Robin*. Penulis menggunakan teknik *smooth service* yang terkenal lebih baik [13]. Oleh karena itu, urutan penjadwalan sesuai dengan *smooth service* untuk kasus ini adalah AABABCABC untuk setiap 1 siklus. Apabila prioritas diatur menjadi sama maka akan menjadi *balanced*.

Singkatnya, penjadwalan *Round Robin* adalah penjadwalan *Weighted Round Robin* yang menganggap semua bobot *Link* sama.

2.6 Transport Layer

Lapisan transport atau *Transport Layer* adalah lapisan keempat dari model referensi jaringan *OSI*. Lapisan ini bertugas untuk mengirim dan menerima data dengan tanpa kesalahan. Pada lapisan ini data dipotong menjadi *segment* lalu dikirimkan ke *Network Layer*. Data akan disusun ulang pada sisi penerima lalu dikirimkan ke *Application Layer*. Berikut layanan yang bekerja di lapisan ini [14]:

a. *Flow Control* untuk menjamin bahwa perangkat yang mentransmisikan data tidak mengirimkan lebih banyak data atau kurang daripada yang dapat ditangani oleh perangkat yang menerimanya.

b. *Packet Sequencing* yang dilakukan untuk mengubah data yang hendak dikirimkan menjadi segmen-segmen data lalu menyusunnya kembali setelah sampai tujuan.

c. Dengan *checksums*, *data integrity* untuk semua *layer* dapat dijamin. Hal ini menjamin data yang dikirim akan sama dengan data yang diterima.

d. *Multiplexing* dan *demultiplexing* yang dapat digunakan untuk menggabungkan data dari bebeberapa sumber untuk mengirimkannya melalui satu jalur data saja.

e. Membuat, mengatur, dan memberhentikan koneksi baik menggunakan *connection oriented* atau *connectionless oriented* dengan protokol *handshake* untuk menjamin koneksi bersifat *robust* sebelum data dikirim.

Contoh protokol yang bekerja pada lapisan ini adalah *TCP* dan *UDP*. Protokol *TCP* (*Transmission Control Protocol)* tersebut bersifat *connection oriented* yang berarti melakukan *three-way handshake* sebelum mengirim data untuk menjamin keutuhan dan *data integrity*. Data akan dipotong-potong menjadi *segment* lalu dikirimkan bersama dengan *checksum* bisa secara urut ataupun tidak ke *host* tujuan. Lalu data akan dibuka dan disusun kembali seperti urutan sebelum dikirim dan dilakukan *error checking* di *host* tujuan. *TCP* juga bersifat *heavy-weight* dan lambat, yang digunakan untuk protokol yang membutuhkan *high reliability* dan waktu pengiriman tidak begitu penting seperti *HTTPS, FTP, SMTP,* dan *Telnet.*

UDP (User Datagram Protocol) bersifat *connection-less oriented* yang berarti tidak perlu melalukan *three-way handshake* untuk memulai, mengatur, dan memberhentikan koneksi. Data akan dikirim langsung ke *host* tujuan tanpa harus meminta persetujuan lalu dipotong menjadi *segment* karena setiap data itu bersifat *independent,* yang menyebabkan data bisa sampai pada tujuan bisa dengan kondisi tidak berurutan dan *unreliable*. *UDP* bersifat *light-weight* dan cepat yang digunakan untuk protokol yang mementingkan kecepatan dan waktu *real-time* seperti *DNS, DHCP, SIP,* dan *RTP*.

2.7 Penelitian Terkait

Berikut adalah beberapa jurnal internasional membahas mengenai *Software Defined Network (SDN)* yang membantu penulis dalam melakukan penelitian :

a. Makalah oleh Christopher Monsanto, dkk. (2013). "*Composing Software-Defined Networks*". Dalam makalah ini dijelaskan kemampuan *SDN* untuk melaksanakan beberapa tugas secara langsung dengan menerapkan *packet-processing rules* pada *switch* dalam sistem jaringan komputer. Itu dicontohkan: *routing, traffic monitoring, acces control, server load balancing*, dan lain sebagainya.

b. Makalah oleh Rahamatullah Khondoker, dkk. (2013). "*Feature-based Comparison and Selection of Software Defined Networking (SDN) Controllers*". Dalam makalah ini, mereka membandingkan antara beberapa *controller SDN* berdasarkan fitur-fitur lalu menganalisis dan menghitung nilai rata rata. Hasilnya: *Ryu Controller* dipilih menjadi yang terbaik dibandingkan *controller* lainnya.

c. Jurnal oleh Yuanhao Zhou, dkk. (2014). "*Method for Load Balancing based on Software Defined Network*". Dalam jurnal mereka ini dijelaskan tentang penelitian terhadap teknologi *SDN* berbasis *OpenFlow* secara singkat. Kemudian mereka melakukan analisa terhadap metode yang ada untuk menerapkan *Load Balancing* pada *SDN*, dan mengemukakan kelebihan dari *SDN*, yaitu lebih efisien dan mudah untuk diterapkan di jaringan komputer.