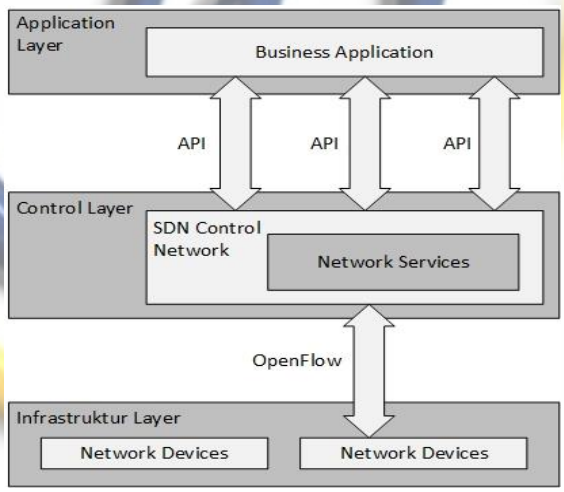
BAB II

LANDASAN TEORI

2.1. *Software Defined Network* (SDN)

Software Defined Network merupakan sebuah konsep baru dalam arsitektur jaringan yang memisahkan *data plane* dengan *control plane* dan dipindahkan secara terpusat pada *controller* yang dapat diprogram. Metode ini memungkinkan administrator jaringan untuk mengontrol kerja perangkat melalui sebuah kontroler tanpa harus mengkonfigurasi tiap perangkat satu persatu. Pada konsep jaringan konvesional dimana *data plane* dan *control plane* terikat erat pada perangkat sehingga konfigurasi *protocol routing* sangat tidak fleksibel, tidak efisien dan konfigurasi dilakukan pada tiap perangkat, hal tersebut tentu tidak dapat memenuhi tuntutan operasional saat ini yang di mana memiliki jaringan besar dan perangkat yang memiliki spesifikasi yang berbeda [7].



Gambar 2.1 *Konsep Arsitektur Jaringan SDN* [7].

Gambar tersebut menjelaskan bahwa sebagian besar kontrol dan protokol jaringan terpusat di perangkat lunak kontrol SDN. Kontroler mengelola semua perangkat jaringan di lapisan infrastruktur dasar yang merupakan *single logical switch virtual.* Operator jaringan dapat mengontrol jaringan melalui *interface* standar (misalnya, *OpenFlow*) secara independen dari vendor perangkat jaringan.

Perangkat-perangkat jaringan hanya dilengkapi dengan fungsi data *forwarding* yang dikendalikan oleh pengontrol SDN. Hal ini membuat desain dan

operasi jaringan menjadi sederhana dan lebih efisien. *Application Programming Interface* (API) antara lapisan aplikasi dan lapisan kontrol dapat memberikan virtualisasi lingkungan jaringan dan sarana untuk menerapkan berbagai kebijakan mengenai *routing*, kontrol akses, rekayasa *traffic*, manajemen daya[7].

Beberapa aspek penting SDN adalah:

1. Adanya pemisahan secara fisik/eksplisit antara *forwarding*/*data-plane* dan *control-plane*.
2. Antarmuka standar (*vendor-agnosic*) untuk memprogram perangkat jaringan.
3. *Control-plane* yang terpusat (secara logika) atau adanya sistem operasi jaringan yang mampu membentuk peta logika (*logical map*) dari seluruh jaringan kemudian merepresentasikannya melalui (sejenis) API.
4. Virtualisasi dimana beberapa sistem operasi jaringan dapat mengontrol bagian-bagian (*slices* atau *substratres*) dari perangkat yang sama

2.2. Protokol Jaringan

Menurut *Mulyanta*, "apabila dua buah sistem saling berkomunikasi, hal yang pertama dibutuhkan adalah kesamaan bahasa yang digunakan, sehingga dapat memahami alur proses komunikasi". Untuk itu, dibutuhkan sebuah mekanisme pengaturan bahasa yang dapat dipahami oleh dua buah sistem tersebut sehingga pertukaran informasi antarsistem akan terjadi dengan benar[9].

Sekumpulan aturan untuk mengatur proses komunikasi ini disebut sebagai protokol komunikasi data. Protokol ini diimplementasikan dalam bentuk program komputer *(software)* yang terdapat pada komputer dan peralatan komunikasi data lainnya. Untuk itu, badan dunia yang menangani masalah standarisasi *IOS (International Organization for Standarization)* membuat aturan baku yang dikenal dengan nama model referensi *OSI (Open System Inerconnection)*

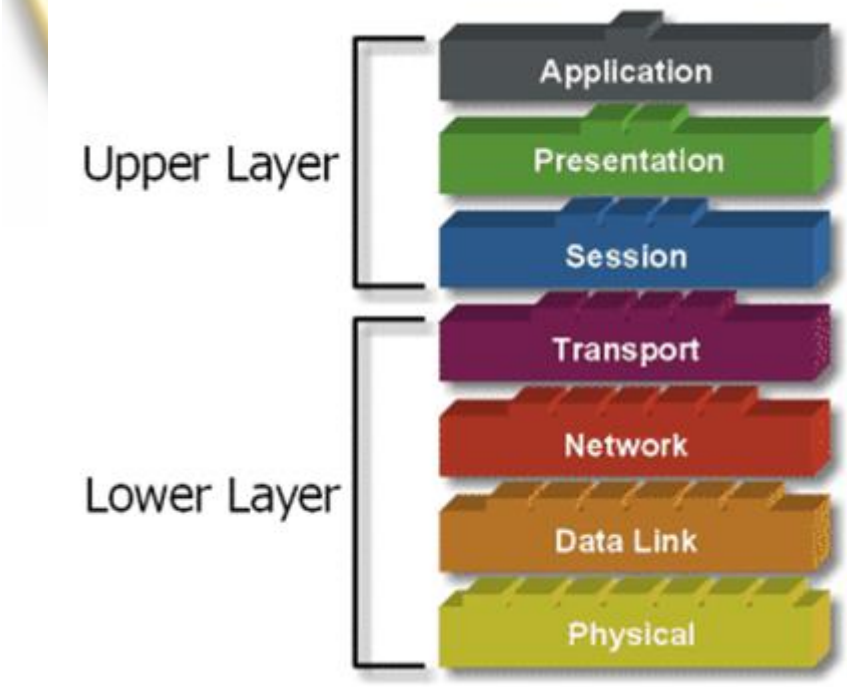
Sedangkan sebuah protokol menjelaskan format dan urutan dari pertukaran pesan-pesan antara dua atau lebih entitas komunikasi dan juga perlakuan apa yang harus diambil pada saat transmisi dan atau saat penerimaan pesan. Protokol

jaringan terbagi atas 3, yaitu model *OSI (Open Systems Interconnection)*, model *TCP/IP*, dan model *ATM (Asynchronous Transfer Mode)*[9].

2.2.1 OSI *Reference Model*

Model OSI terdiri dari tujuh *layer*, yaitu *physical layer*, data *link layer*, *transport layer*, *network layer*, *transport layer*, *session layer*, *presentation layer*, dan *application layer*. Dalam model OSI, peranan *layering* serta proses pertukaran data dalam sistem berlainan akan melalui sebuah hierarki atau tingkat protokol komunikasi[9].

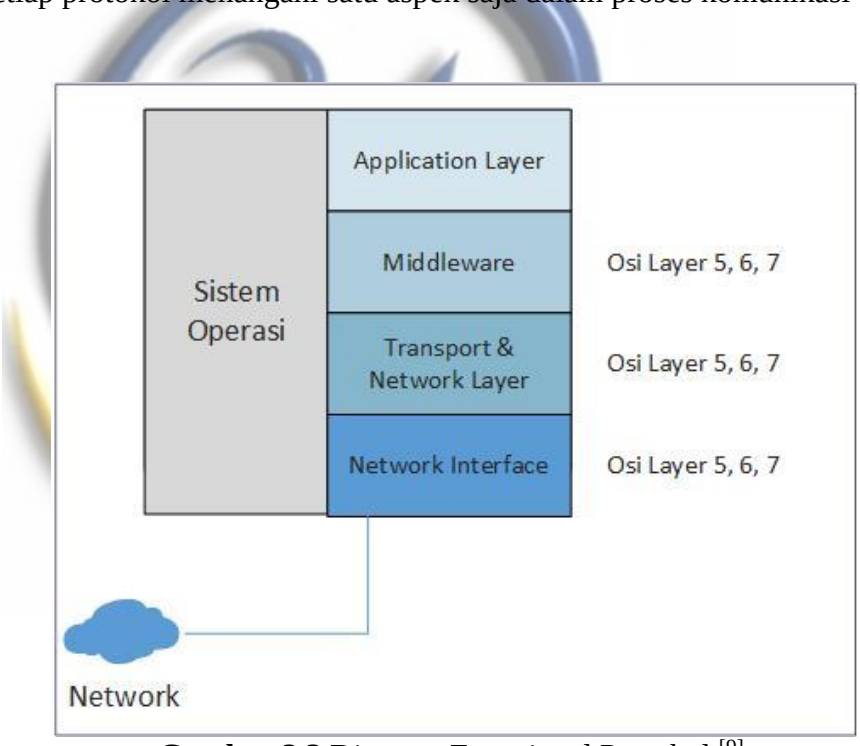
Ketujuh *layer* dari model referensi OSI dapat dibagi dua kategori utama, yaitu *layer* atas dan *layer* bawah. *Layer* atas pada model OSI berkaitan dengan aplikasi dan implementasi secara umum sebuah *software* dengan *layer* tertinggi, yaitu *layer application* merupakan *layer* yang paling dekat dengan pengguna. *Layer* di bawahnya berhubungan dengan penanganan transportasi data. *Physical layer* dan *data link layer* diimplementasikan dalam *hardware* dan *software* sedangkan *layer* lain secara umum diimplementasikan hanya dalam lingkungan *software*. *Layer* yang paling dekat dengan media jaringan adalah *physical layer*. Pembagian kategori pada model referensi OSI ditunjukkan pada gambar 2.3[9].



Gambar 2.2 Pembagian kategori OSI *reference model* [9].

Begitu rumitnya pertukaran data antar jaringan, sehingga desain jaringan komputer harus disederhanakan dalam bentuk bagian-bagian tersendiri dalam lapisan-lapisan *layer* yang mempunyai fungsi dan desain yang berbeda-beda. Setiap *layer* akan memberikan informasi untuk dapat diterapkan dan dipahami oleh *layer* di atasnya [9].

Pertukaran paket antar sistem jaringan akan diperantarai oleh sebuah bahasa komunikasi atau disebut protokol. Protokol tersebut berfungsi antara lain untuk, menentukan jenis konektor, pengalamatan titik-titik komunikasi, identifikasi *interface*, aturan-aturannya, pengaturan aliran data, menjaga ketersediaan data, laporan kesalahan, sinkronisasi, dan lain-lain. Pada praktiknya terdapat beberapa fungsi yang terangkum dalam kumpulan fungsi protokol, dimana setiap protokol menangani satu aspek saja dalam proses komunikasi[9].



Gambar 2.3 Diagram Fungsional Protokol [9].

Pada OSI *reference model*, *layer* bawah didesain untuk berurusan dengan ketersediaan atau memfasilitasi komunikasi dengan *user*. *Layer* ini menyembunyikan detail kerumitan fungsi pertukaran data secara fisik. *Layer* di atasnya (*middleware*) bertanggung jawab terhadap konversi data pada apa yang disebut program aplikasi atau *software*. Secara fungsional terdapat *layer* yang bertugas mengatur transportasi data. *Layer* ini bertanggung jawab terhadap

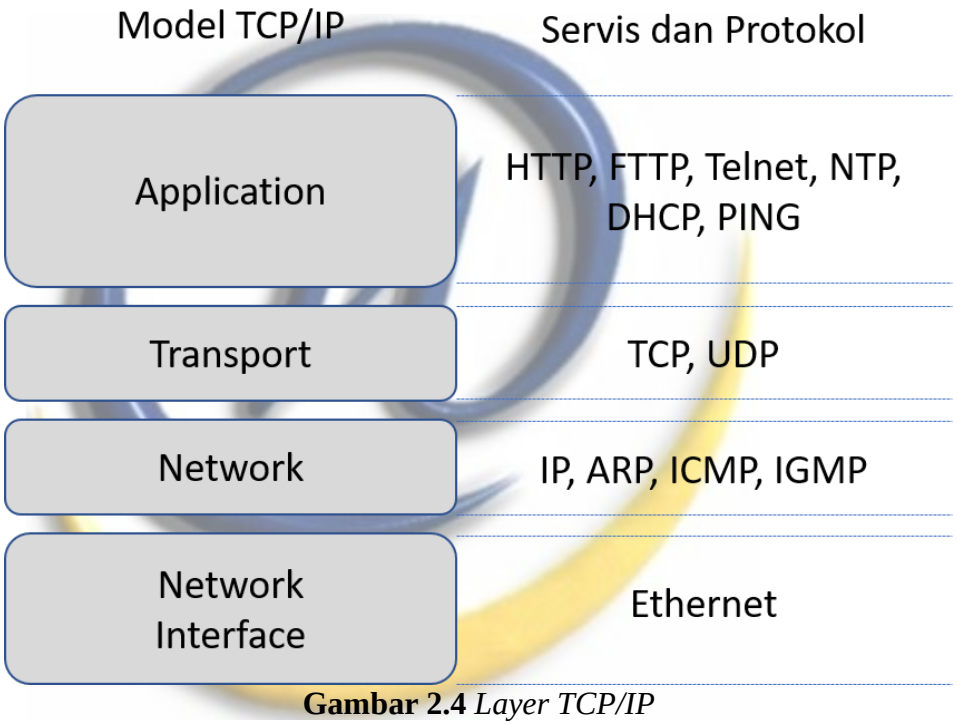
koneksi antar *layer* di atas (*application oriented*) dan *layer* bawah (*network oriented*). Diagram fungsional protokol dapat dilihat pada gambar 2.3[9]. Berikut penjelasan dari ketujuh *layer* pada OSI *reference model*:

Tabel 2.1. OSI *Reference model* [9].

Layer	**Keterangan**
Application	Membuka komunikasi dengan *user* lain dan memberikan layanan seperti *file transfer* ataupun *e-mail* ke *user* lain dalam jaringan. *Layer* ini berfungsi menentukan format data yang akan dipindahkan antar aplikasi dan menyediakan layanan berupa transformasi format data, enkripsi, dan kompresi.
Presentation	Menentukan format data yang dipindahkan di antara aplikasi dan mengelola informasi yang disediakan oleh *application layer* supaya informasi yang dikirim dapat dibaca oleh *application layer* pada sistem lain. *Layer* ini juga menentukan *syntax* yang digunakan antar aplikasi.
Session	Lapisan ini bertanggung jawab atas sesi komunikasi antara berbagai tingkat yang lebih tinggi dalam komunikasi program, proses, dan *user*.
Transport	Berfungsi sebagai pemecah informasi menjadi paket-paket data yang akan dikirim dan menyusun kembali paket-paket data menjadi sebuah informasi yang dapat diterima. Dua protokol umum pada lapisan ini adalah TCP yang berorientasi koneksi dan UDP yang tidak berorientasi koneksi.
Network	*Layer* ini bertanggung jawab dalam *routing* paket data dari *node* sumber ke node tujuan. Selain itu, *layer* ini juga bertanggung jawab dalam pengalamatan *node*, OSPF berada pada *layer* ini.
Data Link	Mengubah paket-paket data dalam bentuk *frame*, menghasilkan alamat fisik, pesan-pesan kesalahan, pemesanan pengiriman data. *Layer* ini juga mengupayakan agar *physical layer* dapat bekerja dengan baik dengan menyediakan layanan untuk mengaktifkan, mempertahankan, dan menonaktifkan hubungan.
Physical	*Physical layer* mentransmisikan data dalam bentuk bit antar *devices* melalui jalur komunikasi dengan cara mengatur karakteristik tinggi tegangan, periode perubahan tegangan, lebar jalur tegangan, jarak maksimum komunikasi dan koneksi.

2.2.2 TCP/IP (*Transmission Control Protocol / Internet Protocol*)

Transmission Control Protocol/Internet Protocol (TCP/IP) adalah satu set aturan standar komunikasi data yang digunakan dalam proses *transfer* data dari satu komputer ke komputer lain di jaringan komputer tanpa melihat perbedaan jenis *hardware*. Protokol TCP/IP dikembangkan dalam riset pertama kali oleh *Defense Advanced Research Projects Agency (DARPA)* di Amerika Serikat dan paling banyak digunakan saat ini yang implementasinya dalam bentuk perangkat lunak (*software*) di sistem operasi[10].



Gambar 2.4 *Layer TCP/IP*

Pada gambar 2.5 dijelaskan bahwa TCP/IP terdiri dari empat lapisan bertingkat yaitu:

1. *Network Acces Layer*
2. *Internet Layer*
3. *Transport Layer*
4. *Application Layer*

Berikut ini tabel penjelasan tentang fungsi dan tugas-tugas keempat layer protokol pada *TCP/IP*:

Tabel 2.2 *Layer* pada TCP/IP

Layer	Keterangan
Application layer	*Layer* ini menangani representasi, *encoding*, serta *control* dialog. Pada *layer* ini terletak semua aplikasi yang menggunakan protokol TCP/IP.
Transport Layer	Berisi protokol yang bertanggung jawab untuk mengadakan komunikasi antara dua *host*/komputer. Kedua protokol tersebut adalah TCP (*Transmission Control Protocol*) dan UDP (*User Datagram Protocol*).
Internet Protocol	*Layer* ini bertanggung jawab dalam proses pengiriman paket ke alamat yang tetap. Pada layer ini terdapat tiga macam protokol, yaitu IP, ARP, dan ICMP.
Network Access Layer	Bertanggung jawab mengirim dan menerima data ke dan dari media fisik berupa kabel, serat optik, atau gelombang radio, sehingga protokol pada layer ini harus mampu menerjemahkan sinyal listrik menjadi data digital yang dimengerti komputer yang berasal dari peralatan lain yang sejenis.

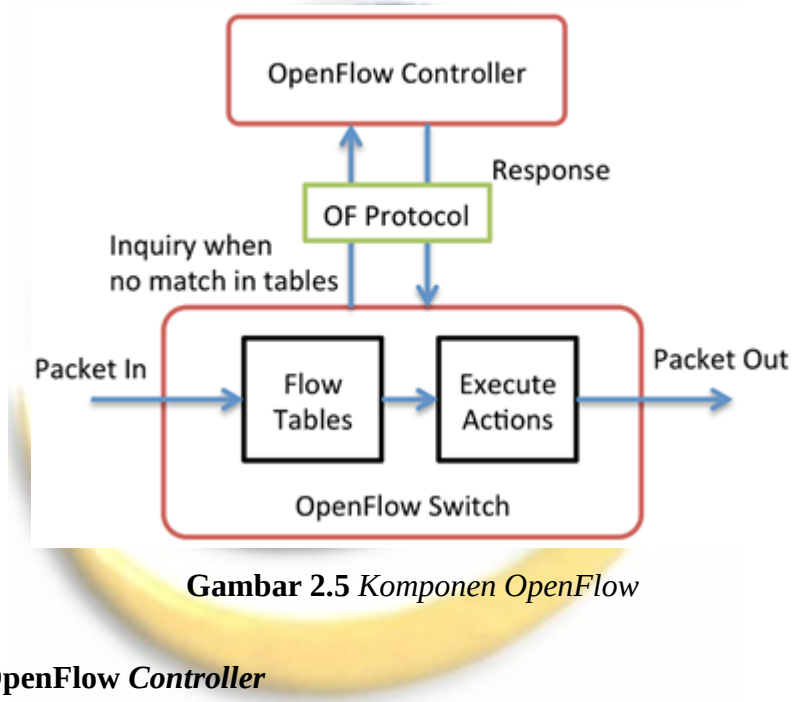
2.2.3 Perbandingan Umum Model OSI dengan TCP/IP

Perbedaan antara model OSI dan model TCP/IP sebagai berikut:

1. Pada model OSI berorientasi pada penyediaan layanan *transfer* data yang *reliable*, sementara TCP/IP memperlakukan *reliability* sebagai masalah *end-to-end*.
2. Setiap *layer* pada model OSI mendeteksi dan menangani kesalahan pada semua data yang dikirim. Pada model OSI *transport layer* memeriksa *reliability* di *source-to-destination*.
3. Pada TCP/IP, kontrol *reliability* dikonsentrasikan pada *Transport Layer*. *Transport Layer* menangani semua kesalahan yang terdeteksi dan memulihkannya. *Layer* ini juga menggunakan *checksum*, *acknowledgement*, dan *timeout* untuk mengontrol transmisi dan menyediakan verifikasi *end-to-end*.

2.3. OpenFlow

OpenFlow adalah, salah satu jenis dari *APIs* (*Application Protocol Interfaces*) dalam jaringan *SDN* yang digunakan untuk mengontrol/mengatur *traffic flows* pada *switch* dalam sebuah jaringan, jadi singkatnya *control plane* berkomunikasi dengan data plane melalui *OpenFlow*. *OpenFlow* dapat bekerja pada *switch* dari berbagai vendor. *OpenFlow* sendiri adalah *southbound interface*, dimana interface yang memungkinkan terjadinya komunikasi antara layer controller dan data plane pada arsitektur *SDN*. *OpenFlow* mempunyai 2 komponen penting yaitu *OpenFlow Controller* dan *Open*VS*witch*. Bisa kita lihat gambar dibawah yang mengambarkan bagian- bagian dari *OpenFlow*[24].



Gambar 2.5 *Komponen OpenFlow*

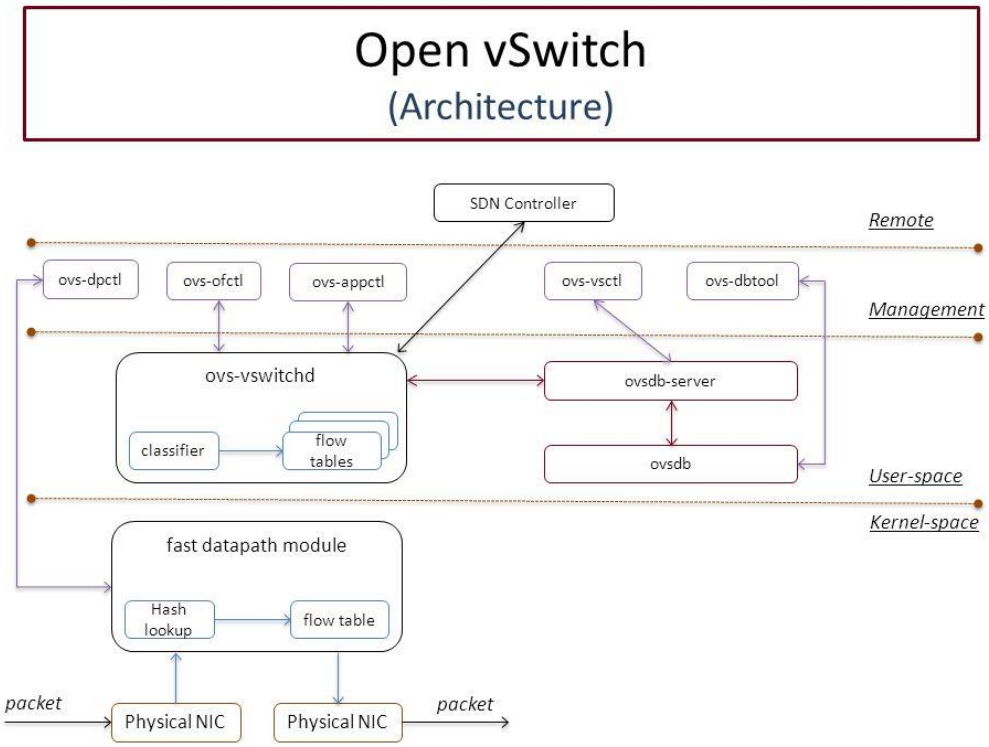
2.3.1. OpenFlow *Controller*

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*).

2.3.2. OpenVSwitch

OpenVSwitch adalah sebuah *switch virtual* berbasis *Linux*. *OpenVSwitch* sendiri merupakan perangkat lunak sumber terbuka yang berada dalam lisensi *Apache 2.0*. *OpenVSwitch* mendukung pengontrolan secara automatis dengan

protokol *OpenFlow*. Open *vSwitch* diinisiasi oleh perusahaan *Nicira*, sebuah perusahaan yang terfokus pada bidang virtualisasi jaringan. *Open vSwitch* mendukung standar manajemen *interface* (seperti *sFlow*, *Netflow*, *RSOAN*, *CLI*). *Open vSwitch* berfungsi sebagai virtual *switch* dalam lingkungan VM yang dirancang untuk mendukung distribusi di beberapa *server* fisik (*physical servers*)[12].



Gambar 2.6 *Open vSwitch*[12].

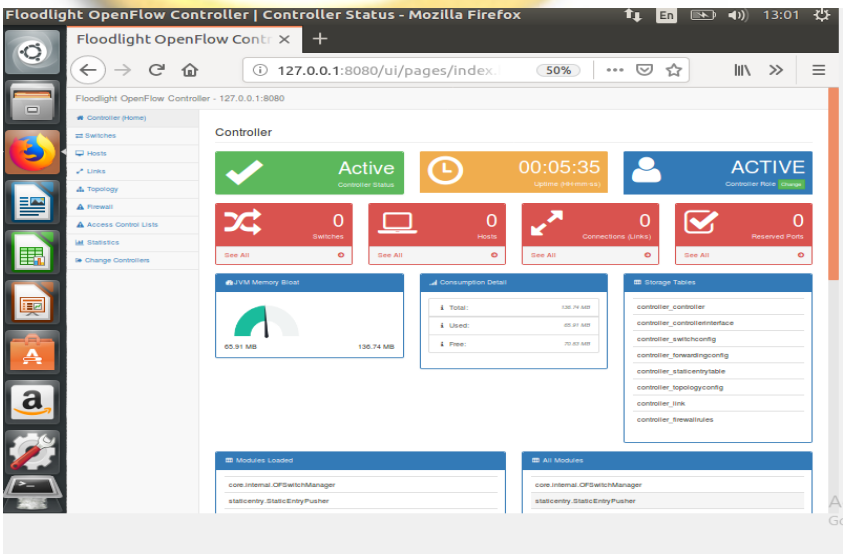
Open vSwitch mendukung beberapa teknologi virtualisasi yang berbasis *Linux* seperti *Xen/Xenserver*, *KVM*, dan *VirtualBox*. Dalam virtual *switch*, diklasifikasikan menjadi dua jalur, *fast path* dan *slow path*. Pada *fast path* terjadi pemrosesan paket-paket data seperti: *packet forwarding*, *trafficking*, dan enkapsulasi paket VLAN. Sedangkan jalur *slow path* difokuskan pada pengelolaan dengan *interface external* [12].

2.4. Floodlight

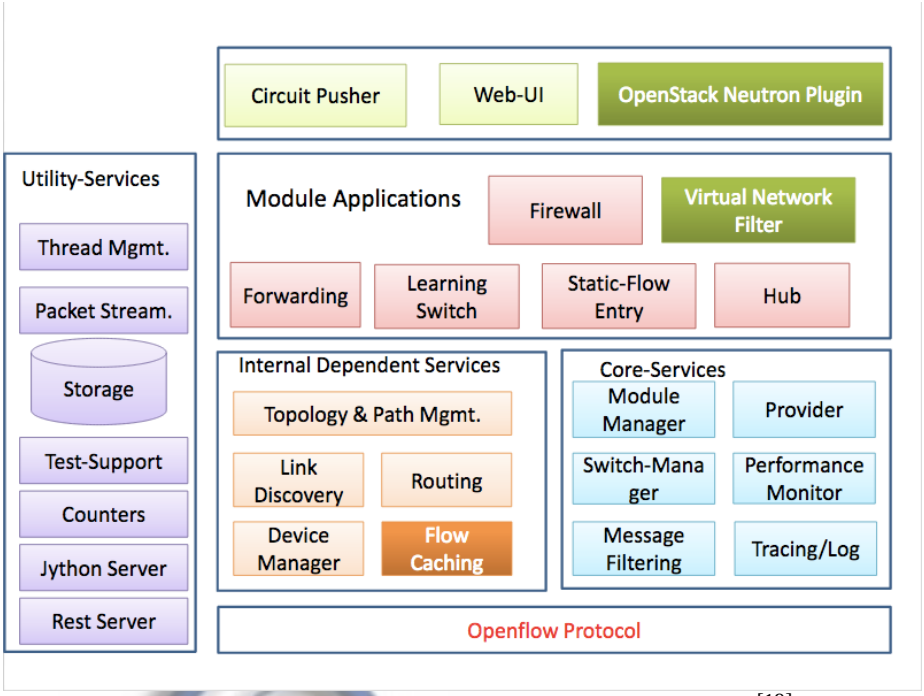
Floodlight adalah *controller Software-Defined Network* terbuka yang merupakan *controller* berlisensi Apache, dan berbasis *Open Flow*. Pengembangan dari controller ini didukung oleh komunitas pengembang termasuk sejumlah insinyur dari Big Switch Networks[19].

OpenFlow adalah protokol terbuka yang dikelola oleh *Open Networking Foundation*. Ini menentukan protokol melalui *switch remote* kontrol dapat memodifikasi perilaku perangkat jaringan melalui didefinisikan dengan baik *"Set instruction forwarding"*. Floodlight dirancang untuk bekerja dengan meningkatnya jumlah *switch*, *router*, *switch virtual*, dan jalur akses yang mendukung protokol *OpenFlow*. Beberapa fitur yang ditawarkan Floodlight sebagai berikut :

a. Menawarkan sistem modul beban yang membuatnya sederhana untukmemperluas dan meningkatkan.
b. Mudah untuk mengatur dengan dependensi minimal.
c. Mendukung berbagai switch *OpenFlow virtual* dan fisik.
d. Dapat menangani *OpenFlow* dan non-*OpenFlow* jaringan campuran dapat mengelola beberapa *" islands"* dari *OpenFlow switch hardware.*
e. Dirancang untuk menjadi kinerja tinggi - adalah inti dari produk komersial dari *Big Beralih Networks*.
f. Dukungan untuk OpenStack (link) platform awan orkestrasi



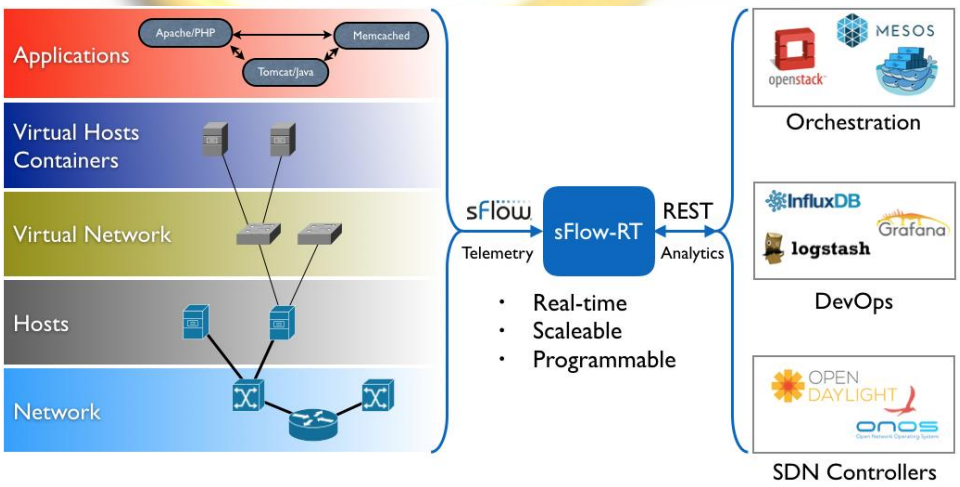
Gambar 2.7 *Web-GUI Floodlight Controller*[19].



Gambar 2.8 *Floodlight Architecture* [19].

2.5. Sflow-RT

Sflow-RT adalah gabungan teknologi analisis sinkron *InMon (Paten AS 8838774, 9509583, 9712443, 9722926)*, yaitu memberikan visibilitas *Real-Time* ke Software Defined Networking (SDN), *DevOps* dan *Orkestrasi Stacks* sehingga memungkinkan kelas baru aplikasi dalam menunjang kinerja seperti penyeimbangan beban , Mitigasi DDoS dan penempatan beban kerja[20].



Gambar 2.9 *Workflow Sflow-RT*[20]

Mesin analisis sFlow-RT menerima aliran telemetri berkelanjutan dari *Agen sFlow* yang tertanam dalam perangkat jaringan, host dan aplikasi dan mengubahnya menjadi metrik yang dapat ditindaklanjuti dan dapat diakses melalui *RESTflow API.*

RESTflow API memudahkan untuk mengkonfigurasi parameter yang disesuaikan, mengambil metrik, menetapkan ambang batas, dan menerima pemberitahuan. Konfigurasi dapat ditulis dalam bahasa apa pun yang mendukung panggilan *HTTP / REST* atau internal menggunakan *JavaScript / ECMAScript sFlow-RT*. Dengan menggabungkan jaringan, host dan pemantauan aplikasi dalam jaringan analitik terintegrasi, sFlow-RT memberikan visibilitas ke dalam aplikasi dan server dan sumber daya jaringan yang diperlukan untuk mempertahankan kinerja.

2.6. Topologi

Topologi menggambarkan metode yang digunakan untuk melakukan pengkabelan secara fisik dari suatu jaringan. Topologi jaringan adalah susunan atau pemetaan interkoneksi antar node, dari suatu jaringan, baik secara fisik (real) dan logis. Jaringan komputer memilki banyak jenis topologi, namun ada 4 (empat) jenis topologi yang umum digunakan adalah Topologi *Bus (Linier),* Topologi *Ring*, Topologi *Tree* dan Topologi *Star*[14]. Dalam penelitian ini, penulis menggunakan jenis Topologi *Tree* untuk melakukan pengujiannya dikarenakan topologi *Tree* merupakan hasil pengembangan dari topologi star dan topologi bus yang terdiri dari kumpulan topologi star dan dihubungkan dengan 1 topologi bus. Kelebihan dari topologi ini juga dalam mendeteksi kerusakan atau kesalahan dalam jaringan dapat membantu penelitian yang dilakukan. Penjelasan lebih lanjut mengenai topologi tree sebagai berikut :

2.6.1. Topologi Tree

. Topologi tree biasanya disebut juga topologi jaringan bertingkat dan digunakan interkoneksi antar sentral[12]. Topologi *tree* disebut juga sebagai topologi jaringan bertingkat. Topologi ini biasanya digunakan untuk interkoneksi antar central dengan *hirarki* yang berbeda.

Topologi *tree* media transmisinya merupakan satu kabel yang bercabang namun *loop* tidak tertutup. Topologi *tree* dimulai dari suatu titik yang disebut *"headend".* Dari *headend* beberapa kabel ditarik menjadi cabang, dan pada setiap cabang terhubung beberapa terminal dalam bentuk bus, atau dicabang lagi hingga menjadi rumit[14].

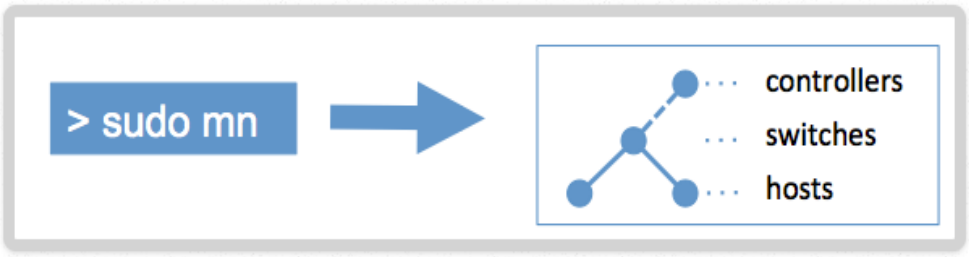
Topologi *tree* digunakan untuk menghubungkan beberapa *LAN (Local Area Connection)* dengan *LAN* lain. Hubungan antara LAN dilakukan via *hub*. Masing-masing *hub* dapat dianggap sebagai akar dari masing-masing pohon *tree*[14].

Pada topologi *tree* terdapat kelebihan dan kekurangan antara lain sebagai berikut :

1. Kelebihan topologi *Tree :*
   a. Mudah mengembangkan menjadi jaringan yang lebih luas
   b. Mudah mendeteksi kerusakan atau kesalahan.
   c. Manajemen data yang baik.
   d. Manajemen mudah karena mudah melakukan identifikasi dan isolasi kesalahan dalam jaringan.
2. Kekurangan topologi *Tree* :
   a. Kinerja yang lambat
   b. *Hub* menjadi peran penting.
   c. Menggunakan biaya yangbanyak karena menggunakan banyak kabel dan *hub.*

2.7. Mininet

Mininet adalah suatu *software* emulator yang memungkinkan untuk melakukan *prototyping* pada jaringan yang luas dengan hanya menggunakan satu mesin. Pada *mininet* ini dilakukan perancangan jaringan dengan topologi yang diinginkan. Secara sederhana mininet ini berfungsi untuk emulasi pada bagian *data path* untuk mengetes konfigurasi jaringan SDN. Sedangkan untuk melakukan testing pada mininet dapat dilakukan dengan *command "sudo mn".* Dengan *command* ini *mininet* akan mengemulasikan konfigurasi jaringan SDN yang terdiri dari 1 *controller*, 1 *switch* dan 2 *host*[15].



Gambar 2.10 *Single command Mininet* [15].

Mininet mendukung penelitian, pengembangan, pembelajaran, pembuatan *prototipe*, pengujian, dan *debugging* yang dapat diambil manfaat dari memiliki jaringan eksperimental yang lengkap pada *laptop* atau *PC* lainnya. *Mininet* menyediakan *testbed* jaringan yang sederhana dan murah untuk mengembangkan aplikasi *OpenFlow* serta memungkinkan beberapa pengembang bersamaan untuk bekerja sendiri pada topologi yang sama. Mendukung *tes regresi* tingkat sistem yang dapat diulang, dapat mengaktifkan pengujian topologi yang kompleks , tanpa perlu memasang jaringan secara fisik termasuk *CLI* pada topologi dan *OpenFlow* untuk *debugging* atau menjalankan tes di seluruh jaringan.

Mininet mendukung topologi khusus yang dibuat secara manual dan termasuk topologi *parametrized* yang menyediakan *API Python* yang mudah dan dapat diperluas untuk pembuatan dan eksperimen. Jaringan *Mininet* menjalankan *coding* pada aplikasi jaringan *Unix* atau *Linux* standar serta *kernel Linux* asli termasuk ekstensi *kernel* apa pun yang memungkinan asalkan kompatibel jaringan yang akan dibangun[15].

Karena itu, *coding* yang akan dikembangkan dan uji pada *Mininet*, *OpenFlow*, *switch*, atau *host*, dapat pindah ke sistem nyata dengan melakukan perubahan seminimal mungkin untuk pengujian dunia nyata dan evaluasi kinerja. Yang harus diingat bahwa desain yang bekerja di *Mininet* biasanya dapat langsung pindah ke *hardware switch* asli untuk dilakukan *line-rate packet forwading*.

2.8. *Oracle VM VirtualBox*

Oracle VM VirtualBox adalah sebuah perangkat lunak virtualisasi, yang dapat digunakan untuk mengeksekusi sistem operasi tambahan di dalam sistem operasi utama. Sebagai contoh, jika seseorang mempunyai sistem operasi

Windows yang terpasang pada komputernya, maka seseorang tersebut dapat pula menjalankan sistem operasi lain yang diinginkan di dalam sistem operasi *Windows* tersebut [16].

Fungsi ini sangat penting jika seseorang ingin melakukan ujicoba dan simulasi instalasi suatu sistem tanpa harus kehilangan sistem yang ada. Aplikasi dengan fungsi sejenis *VirtualBox* lainnya adalah *VMware* dan *Microsoft Virtual PC*. [16]

2.9. QoS (*Quality of Service*)

Quality of Service (QoS) didefinisikan sebagai suatu pengukuran tentang seberapa baik jaringan dan merupakan suatu usaha untuk mendefinisikan karakteristik dan sifat dari suatu layanan. Pada jaringan berbasis IP, IP QoS mengacu pada performansi dari paket-paket IP yang lewat melalui satu atau lebih jaringan. *QoS* didesain untuk membantu *end user* menjadi lebih produktif dengan memastikan bahwa *end user* mendapatkan performansi yang handal dari aplikasi aplikasi berbasis jaringan[17].

2.9.1 Parameter QoS

Faktor-faktor yang mempengaruhi performa *Quality of Service* pada *routing* protokol menurut beberapa ahli adalah sebagai berikut:

1. *Throughput*
2. *Delay*
3. *Jitter*
4. *Packet Loss*

a. *Throughput*

Throughput adalah kecepatan (*rate*) transfer data efektif, yang diukur dalam *bps*. *Throughput* merupakan jumlah total kedatangan paket yang sukses yang diamati pada tujuan selama interval waktu tertentu, dibagi oleh durasi interval waktu tersebut. Nilai *throughput* dapat dihitung menggunakan persamaan sebagai berikut:

$$Throughput = \sum_{t=Tt}^{t=Tt+1} Pt \ , 0 \ \leq t \ \leq T \ldots\ldots\ldots(1)$$

Contoh perhitungan berdasarkan rumus *Throughput*:

$$552 + 552 + 552 = \frac{1652}{12} = 138\ ms \ldots\ldots\ldots\ldots\ldots\ldots (2)$$

Tabel 2.3 Keterangan *Throughput*.

Angka	Keterangan
552 + 552 + 552	Paket data diterima simbol (+) (ditambahkan semua paket data di terima)
1652	Hasil dari penambahan paket data diterima.
12	Waktu perjalanan simulasi
138 ms	Hasil bagi dari paket data diterima dan waktu simulasi.

Dimana:

P = Besar paket yang diterima (bit)
T = Waktu Simulasi (detik)
T = Waktu pengambilan sampel (detik)

Klasifikasi throughput menurut versi *TIPHON* (*Telecomumunications and Internet Protocol Harmonization Over Networks*) .

Tabel 2.4 Klasifikasi *Throughput*

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

(Sumber: TIPHON)

b. *Delay* (*Latency*)

Delay adalah selang waktu yang dibutuhkan oleh suatu paket data saat data mulai dikirim dan keluar dari proses antrian sampai mencapai tujuan. *Delay* dapat dipengaruhi oleh jarak, media fisik, kongesti atau juga waktu proses yang lama. *Delay* dapat dihitung dengan menggunakan persamaan berikut:

$$Delay = \frac{Tr - Ts}{Pr} \times 100\ ,0\ \leq t\ \leq T \ldots\ldots\ldots\ldots\ldots\ldots (1)$$

Contoh perhitungan berdasarkan rumus *Delay*

$$Delay: \frac{10}{12} = \frac{0.83}{15} = 0.55 \times 100 = 5.54\ ms \ldots\ldots\ldots\ldots (2)$$

Tabel 2.5 Keterangan *delay*

Angka	Keterangan
10	Jumlah penerima packet data (simbol (R)) total paket data.
12	Waktu simulasi
0.83	Hasil bagi dari penerima packet data (Total) dengan waktu simulasi
15	Jumlah penerima packet data (symbol (-))
0.55	Hasil bagi packet data (R) dan packet data (-).

Dimana:

Tr = Jumlah waktu penerimaan paket data (detik)
Ts = Jumlah waktu penngiriman paket data (detik)
Pr = Jumlah paket data yang diterima selama satu detik

Nilai *delay* (latency) sesuai dengan versi *TIPHON* (*Telecomumunications and Internet Protocol Harmonization Over Networks*) [18].

Table 2.6 Klasifikasi *delay* (*latency*)

Kategory Latensi	Besar *Delay*	Indeks
Sangat bagus	< 150 ms	4
Bagus	150 s/d 300	3

(Sumber: TIPHON)

c. *Jitter*

Jitter di definisikan sebagai variasi *delay* yang diakibatkan oleh Panjang *queue* dalam suatu pengolahan data dan *reassemble* paket paket data di akhir pengiriman akibat kegagalan sebelumnya. Secara umum *jitter* merupakan masalah dalam *slow speed links*. Jitter dapat dilihat dari rumus di bawah ini :

Persamaan perhitungan *jitter*: $Jitter = \frac{Total\ variasi\ delay}{Total\ paket\ yang\ diterima} \ldots\ldots (1)$

Total variasi delay diperoleh dari:

Total variasi delay = Delay – Rata-rata Delay……..(2)

Contoh perhitungan berdasarkan rumus *jitter*:

Jitter: $\frac{5.54}{12} = 0.46$ ($5.54 - 0.46 = \frac{5.08}{10} = 0.508$)……..(3)

Tabel 2.7 Keterangan *jitter*

Angka	Keterangan
5.54	Hasil *delay*
12	Waktu simulasi
5.08	Hasil dari *delay* di kurang dengan hasil *delay* di bagi waktu simulasi
10	Jumlah menerima pakket data (simbol (-))

Terdapat empat kategori penurunan performansi jaringan berdasarkan nilai *peak jitter* sesuai dengan versi *TIPHON (Telecomumunications and Internet Protocol Harmonization Over Networks)*[18].

Tabel 2.8 Ukuran kualitas *jitter*

Kategori degredasi	Peak Jitter (ms)	Indeks
Sangat bagus	0	4
Bagus	0 s/d 75 ms	3
Sedang	75 s/d 125 ms	2
Jelek	125 s/d 225ms	1

(Sumber: TIPHON)

d. *Packet Loss*

Packet Loss adalah suatu parameter yang menggambarkan suatu kondisi yang menunjukkan jumlah total paket yang hilang atau rusak selama pengiriman paket data berlangsung. *Packet loss* dapat terjadi karena *collision* dan *congestion* pada jaringan. *Packet loss* dapat dihitung dengan menggunakan persamaan sebagai berikut:

$$Packet\ Loss = \frac{Pd}{Ps} \times 100\ , 0\ < t\ < T \text{.....(1)}$$

Contoh perhitungan berdasarkan rumus *packet loss*:

$$\text{Packet Loss} = \frac{18}{5} = \frac{3.6}{12} = 0.3\ \times 100 = 30\ ms \text{.....(2)}$$

Tabel 2.9 Keterangan *packet loss* [17].

Angka	Keterangan
18	Paket data *drop* selama 1 detik (Simbol (D))
5	Jumlah packet data.
0.3	Hasil bagi paket data *drop* 1 detik dengan jumlah paket data
100	Mendapatkan Hasil

Dimana :

Pd = Jumlah paket data yang mengalamai *drop* selama satu detik (paket)
Ps = Jumlah paket data yang dikirim selama satu detik (paket)
T = Waktu Simulasi (detik)
T = Waktu pengambilan sampel (detik)

Nilai *packet loss* sesuai dengan versi *TIPHON* (*Telecomumunications and Internet Protocol Harmonization Over Networks*) [18].

Tabel 2.10 Ukuran kualitas *packet loss*.

Kategori Degredasi	*Packet Loss*	Indeks
Sangat Bagus	0 %	4
Bagus	3 %	3
Sedang	15 %	2
Jelek	25 %	1

(Sumber: TIPHON)

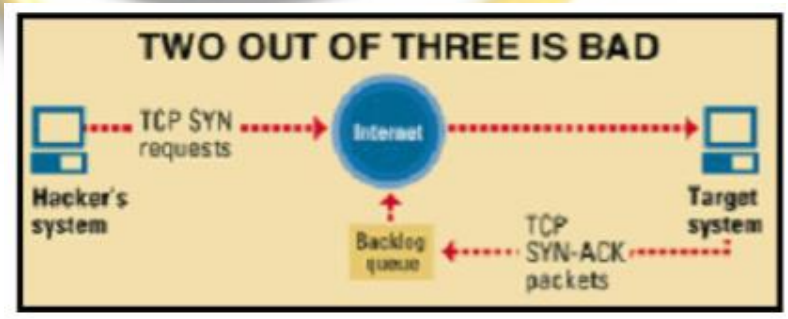
2.10 *Distributed Denial of Services* (*DDoS*)

Distributed Denial of Service (DDoS) adalah jenis serangan yang dilakukan secara masif dengan tujuan mengganggu hak akses pengguna jaringan. *DDoS* merupakan serangan *flooding trafic* yang dilakukan dengan sengaja utuk menggangu *QoS* dari sistem jaringan yang bertujuan untuk membuat sumber daya server habis. Serangan *DDoS* pada dasarnya sama dengan *Denial of Service (DoS)* namun serangan dilakukan dengan banyak sumber secara serentak[19]. Untuk melancarkan serangan *DDoS*, penyerang biasanya mengumpulkan pasukan dengan cara mengambil alih komputer-komputer yang kemudian dijadikan *zombie* untuk di perintah dan dikendalikan oleh *bootnet* [21].

Sebuah usaha serangan untuk membuat komputer atau *server* tidak bisa bekerja dengan baik. Dalam hal dapat menyebabkan performa *server* komputer menjadi sangat lambat. Hal itu dikarenakan ada ribuan *Spam* sistem yang menyerang secara bersamaan.

2.11 SYN *Flooding*

SYN Flooding merupakan network *Denial Of Service* yang memanfaatkan *'loophole'* pada saat koneksi *TCP/IP* terbentuk. Kernel Linux terbaru telah mempunyai option konfigurasi untuk mencegah DOS dengan mencegah atau menolak *cracker* untuk mengakses sistem[22].



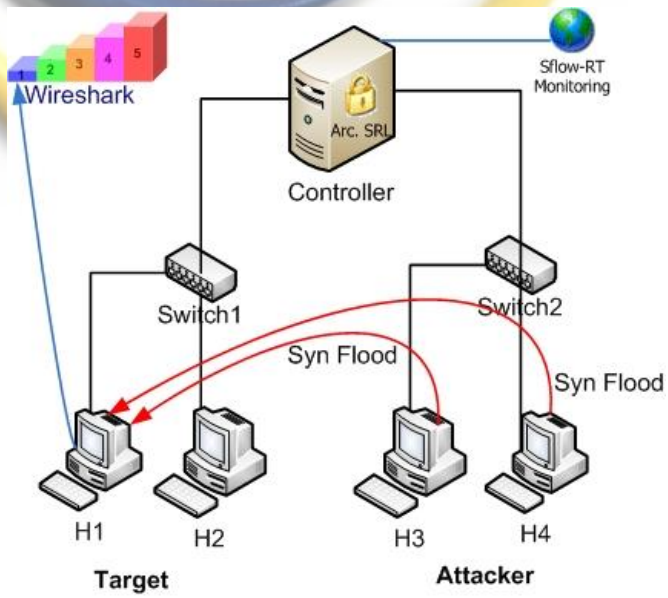
Gambar 2.11 Mekanisme Serangan *SYN Flooding*[22]

Pada kondisi normal, client akan mengirimkan paket data berupa *SYN (synchronization)* untuk mengsinkronkan pada *server*. Lalu *server* akan menerima request dari client dan akan memberikan jawaban ke *client* berupa *ACK* (*Acknowledgement)*. Sebagai tanda bahwa transaksi sudah dimulai (pengiriman dan penerimaan data), maka *client* akan mengirimkan kembali sebuah paket yang berupa *SYN* lagi. Jenis serangan ini akan membanjiri *server* dengan banyak paket

SYN[20]. Karena setiap pengiriman paket *SYN* oleh *client*, *server* pasti akan membalasnya dengan mengirim paket *SYN ACK* ke *client*. *Server* akan terus mencatat dan membuat antrian *backlog* untuk menunggu respon *ACK* dari *client* yang sudah mengirim paket *SYN* tadi. Biasanya memori yang disediakan untuk *backlog* sangat kecil. Pada saat antrian *backlog* ini penuh, sistem tidak akan merespond paket *TCP SYN* lain yang masuk dalam bahasa sederhananya sistem tampak *hang*. Sialnya paket *TCP SYN ACK* yang masuk antrian *backlog* hanya akan dibuang dari *backlog* pada saat terjadi *timeout* dari *timer TCP* yang menandakan tidak ada respon dari pengirim.

2.12 *Architecture* SRL *(Self Reproducing Loops)*

Architecture SRL adalah kerangka kerja pemantauan paket berbasis OpenFlow yang diimplementasikan pada sisi pengontrol. Setiap kali suatu paket datang secara bergantian yang entri tidak cocok dari entri tabel maka akan diteruskan ke *controller* lalu akan menghitung *rute* dan menginstal *entri tabel flow* masuk pada *switch* untuk meneruskan paket. Penyerang dapat mengambil keuntungan dari situasi ini dan dapat mengirim sejumlah besar paket *syn* yang entrinya tidak akan cocok dengan arus tabel dan kemudian *controller* akan menginstal aturan baru untuk penyerang.

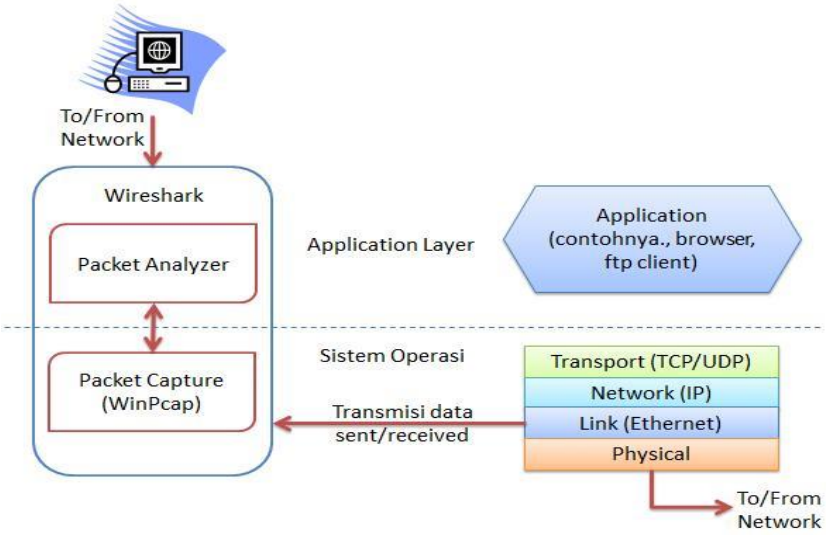


Gambar 2.12 *Architecture* SRL

Karena itu, mungkin saja terjadi bahwa semua entri asli mungkin diganti entri yang tidak sah dan *tabel flow* akan diisi aturan *flow* yang tidak perlu. Ini menyebabkan *tabel flow* beralih atau *overflow* yang pada gilirannya mengarah ke *(Ternary Content Addressable Memori)* agar mengalami *TCAM crash* atau *down*. *SRL* mengatasi hal ini dengan cara menginstal sejumlah entri flow dalam *controller*. Jadi ketika seorang penyerang mencoba mengirim sejumlah besar paket *SYN* dan jika jumlah *flow* semakin terinstal lebih dari batas tetap maka semua entri diblokir. Keuntungan dari SRL adalah bahwa SRL memblokir bahkan setelah penyerang menyelesaikan koneksi *TCP*. Selain itu, SRL juga tidak gunakan entri sementara apa pun untuk menginstal aturan flow melainkan untuk mencapai tujuan ini menggunakan *SRL* menggunakan modul fungsional dalam arsitektur pengontrol yang ada pada *Controller Floodlight*[6].

2.13 Wireshark

Wireshark adalah program *Network Protocol Analyzer* alias penganalisa protokol jaringan yang lengkap. Program ini dapat merakam semua paket yang lewat serta menyeleksi dan menampilkan data tersebut sedetail mungkin, misalnya postingan komentar kamu di blog atau bahkan *Username dan Password.* Fungsi utama Wireshark dibuat untuk Administrator Jaringan untuk dapat melacak apa yang terjadi didalam jaringan atau untuk memastikan jaringannya bekerja dengan baik [23]. Di dalam penelitian ini, penulis menggunakan *Wireshark* sebagai program aplikasi tambahan pembantu untuk merekam data *traffic* yang berjalan ketika pengujian sedang berlangsung. Data tersebut selanjutnya akan digunakan penulis sebagai analisa perhitugan *QoS.*



Gambar 2.13 Mekanisme Kerja *Wireshark*[23]

Cara kerja *wireshark* sebagian besar terdiri dari dua tahapan:

1. Merekam semua paket yang melewati *interface* yang dipilih *(Interface* adalah perangkat penghubung antar jaringan, bisa melalui *wifi* atau *ethernet* atau *lan card).*
2. Hasil rekaman tadi dapat dianalisa. disini kita dapat memfilter protocol apa yang kita inginkan seperti *tcp, http, udp* dan sebagainya. *Wireshark* juga dapat mencatat *cookie, post dan request*.