BAB 2
DASAR TEORI

2.1 KAJIAN PUSTAKA

Pada penelitian [2] menjelaskan sebuah arsitektur jaringan linier dengan menggunakan 8 *switch*, 160 *host* sehingga 1 *switch* mengelola 20 *host*. Pada penelitian tersebut bertujuan untuk mengetahui performa *controller Opendaylight* dan *controller Ryu* dengan melihat pada nilai *troughput, delay, paket loss* yang menggunakan simulator *D-ITG* serta mengetahui perbandingan *resourch utilization* menggunakan *phoronix test- suite*, dengan mensimulasikan pengiriman data dengan nilai beban *traffic* yang berbeda yaitu 100Mb, 500Mb, 1Gb hingga 10 Gb pada sejumlah *switch* dan *host.* Hasil dari penelitian ini *controller OpenDayLight* memiliki performa yang lebih baik dibandingkan dengan *controller Ryu*. Hasil pada *controller ryu* rata-rata nilai *troughput* sebesar 325,682 Mb/s, rata-rata nilai *delay* sebesar 0,31335s dan untuk nilai rata-rata *packet loss* sebesar 4,59% sedangkan untuk hasil penelitian pada *controller OpenDayLight* menghasilkan nilai rata rata *troughput* sebesar 318,749 Mb/s, rata-rata nilai *delay* sebesar 0,622309s dan untuk nilai rata-rata *packet loss* sebesar 10,11% lalu pengujian masing-masing percobaan sebanyak 10 kali. Hasil percobaan pada pengujian *Reasource Utilization* pada *controller OpenDayLight* memiliki hasil yang lebih baik dibandingkan dengan *controller Ryu*, hasil itu dapat dilihat dari performa *CPU* dan *Memory*.

Pada penelitian [13] membahas tentang parameter rata-rata *troughput TCP/UDP*, rata-rata *bandwidth*, *packet loss*, *latency, topology discovery time* dan *prediction impection*. Pada penelitian ini menggunakan beberapa skenario dalam pengujian, skenario yang diterapkan berupa perubahan pada topologi jaringan yaitu menggunakan topologi *linear*, topologi *tree* yang terdiri dari 7 *switch* dan 12 *host* dan topologi *custom* yang terdiri dari 7 *switch*, 7 *host* dan 3 *controller*. Pada semua topologi jaringan memiliki *bandwidth* link 1.000 *Mbit/s*, pengujian ini juga mengui *controller* dengan nilai *suggestion window* yang berbeda yaitu sebesar 2 *Mb*, 20 *Mb* dan 32 *Mb*. Hasil dari uji coba penelitian ini adalah pada pengujian *troughput* bahwa *controller POX* dan *NOX* memiliki *troughput* terbaik di semua jenis topologi yang

diuji dibandingkan dengan *controller FloodLight*. Pada pengujian *bandwidth UDP*, *controller POX* dan *NOX* menghasilkan nilai yang lebih baik dibandingkan dengan *controller FloodLight*. Pada pengujian *Latency, controller POX* menjadi *controller* terbaik dengan respons tertinggi per milidetik dan pengujian terakhir yaitu *topology discovery* yang menghasilkan bahwa *controller Floodlight* tercatat pada semua jenis topologi.

Pada penelitian [4] menggunakan Topologi *Full-Mesh* mengenai pengujian kualitas jaringan pada SDN menggunakan jenis *controller RouteFlow* dan arsitektur jaringan *Full Mesh* yang terdiri dari empat buah *switch*, empat buah komputer dan satu buah *controller*. Pengujian untuk kualitas jaringan SDN, menggunakan dua skenario pengujian yaitu menggunakan trafik data dan tanpa menggunakan trafik data dimana terdapat dua protokol layer *transport* (*TCP* dan *UDP*) digunakan untuk mengirimkan data. Pada pengujian ini menggunakan skema pertukaran data dengan ukuran yang bervariasi yaitu sebesar 4,8 *Mbyte*, 6,4 *Mbyte*, 8 *Mbyte*, 9,6 *Mbyte*, 11,2 *Mbyte,* dan 12,8 *Mbyte*. Parameter *QoS* yang diuji yaitu parameter *delay*, *jitter* dan *troughput* selain menguji parameter QoS terdapat juga parameter lain yang diuji yaitu melakukan pengukuran waktu konvergensi jaringan. Hasil dari pengujian ini adalah penggunaan menggunakan protokol *UDP* menghasilkan nilai parameter yang lebih baik dibandingkan dengan protokol *TCP*, untuk waktu konvergensi sebuah arsitektur jaringan dapat dikategorikan dalam keadaan konvergen sebesar 12,18 ms.

Pada penelitian [9] yang menganalisa parameter *QoS* berupa *troughput* dengan menggunakan *Iperf*. Arsitektur jaringan yang digunakan yaitu topologi *Mesh* yang memiliki 6 *switch* dimana 5 *switch* mengontrol 10 *host* dan 1 *switch* lainnya dihubungkan dengan 1 *controller* sehingga pada arsitektur tersebut terdapat 50 *host*. Pengujian dilakukan sebanyak 6 kali dengan skenario penelitian setiap *switch* mengirimkan paket dengan mengubah jumlah *host* disetiap pengujian. Hasil dari pengujian ini disetiap percobaan menunjukan bahwa semakin banyak jumlah *switch* dan *host*, semakin menurun nilai *troughput*.

Penelitian [10] menguji parameter *QoS* berupa *delay*, *jitter, paket loss* dan *troughput* pengujian lainnya yaitu *resource utilization*. Desain arsitektur yang digunakan adalah topologi *tree* dengan tiga rancangan yang terdiri atas *switch* tujuh,

sembilan dan sebelas. Pengujian tersebut memerlukan *iperf* dalam memberikan *background traffic*, *background traffic* yang diberikan yaitu 0 *Mbps*, 50 *Mbps*, 100 *Mbps*, 150 *Mbps* dan 200 *Mbps*. Nilai parameter yang telah didapat akan dibandingkan dengan nilai yang telah ditetapkan oleh badan standarisasi *ITU-T*.

Pada penelitian [1] menggunakan topologi *custom* yang tediri dari 18 *host* dan 10 *switch* dengan skenario pengujian pengiriman paket *TCP* dan *UDP* menggunakan *Iperf 3*. Pada penelitian tersebut telah diperoleh nilai *jitter, packet loss, troughput UDP, troughput UTP* serta *bandwidth* rata rata di setiap *host* yang terdapat pada topologi ini. Lalu untuk analisa terhadap *device Openflow Switch* diperoleh data seperti *Packet Received, Packet Send, Byte Received* dan *Byte Send*.

Penelitian [8] melakukan pengujian *QoS* pada parameter *troughput* serta *latency*. Pengujian tersebut menggunakan simulator *Cbench* dengan menggunakan jumlah *switch* dan *host* yang bervariasi yaitu 200 *switch* dan 400 *host*, 40 *switch* dan 800 *host*, 60 *switch* dan 1600 *host*, 80 *switch* dan 3200 *host,* 100 *switch* dan 6400 *host*, 12 *switch* dan 12.800 *host.* Pengujian ini diperoleh hasil untuk penanganan aliran data yang berfungsi mengolah data dalam jumlah yang besar, *controller maestro* memiliki kemampuan lebih baik dibandingkan dengan *controller Floodlight, Ryu, POX* dan *ONOS*. *Controller maestro* memiliki selisih nilai *troughput* 500 hingga 4.000 *Flow/s* dan nilai *latency* lebih baik dibandingkan *controller* lain dengan selisih nilai 1.000 hingga 5.000 *ms*.

Penelitian [11] menggunakan jenis topologi *linier*, ini dikarenakan topologi *linier* tidak perlu membuat topologi secara manual yang disebut dengan topologi *custom* atau topologi standar seperti topologi *single* dan topologi *tree.* Topologi liner digunakan untuk menguji nilai *troughput*, *delay* dan *controller libfluid, ONOS, OpenDayLight, POX* dan *ryu* dalam bekerja menggunakan *Iperf.* Topologi *liner* terdiri berbagai jumlah *switch*, untuk skenario yang dijalankan menggunakan *switch* sebanyak 8, 16, 32, 64, 128, 256, 512 dan 1.024. Hasil dari nilai *troughput* yaitu semakin meningkat beban kerja nilai *troughput* pada *controller* semakin menurun, pengujian ini dilakukan hingga *controller* berhenti merespon, untuk *controller libfuid* dan *POX* berhenti merespon ketika jumlah *switch* sebanyak 1.024 *switch* dan *controller* lainnya berhenti ketika jumlah *switch* sebanyak 512 *switch* untuk nilai *troughput* terendah dimiliki oleh *controller OpenDayLight*. Pada nilai

delay dilihat dari rata-rata *Round Trip Time (RTT) controller* meningkat sebanding dengan bertambahnya jumlah *switch*, untuk *controller ONOS, OpenDayLight, POX* dan *Ryu* memiliki nilai *delay* yang sama dengan *overload* beban jaringan kurang dari 4ms. Untuk *controller libfluid* meningkat hingga mencapai 715,73 ms pada 512 *switch*. *Controller ONOS* memiliki nilai *delay* yang palig rendah dan *controller libfluid* memiliki *delay* tertinggi.

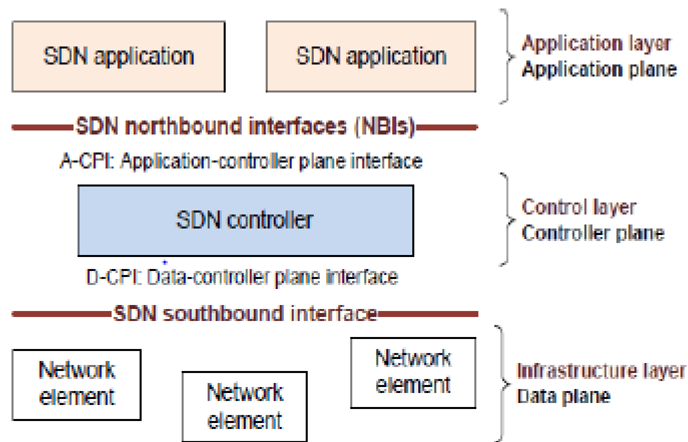
Pada penelitian [12] mengimplementasikan *controller Ryu, POX* dan *Pyretic openflow* dengan basis menggunakan bahasa *Pyton*. Penelitian ini mensimulasikan menggunakan jenis topologi *tree* dengan metode virtualisasi berbasis proses dan mekanisme *namespace* jaringan. Hasil uji performansi *controller* tersebut didapatkan bahwa *controller POX* memiliki *Round Trip Time (RTT)* rata-rata terbesar 0,185 ms dan nilai *RTT* minimum terbesar yaitu 0,145 ms. *Controller pyretic* memiliki nilai minimum sebesar 0,137 ms, nilai maksimum sebesar 0,191 ms dan *RTT* rata-rata sebesar 0,161 ms. Controller Ryu memiliki nilai *RTT* maksimum sebesar 0,175 ms. Hasil yang didapat bahwa *Controller Pyretic* menunjukkan performa yang lebih baik dibandingkan dengan *controller Ryu* dan *controller POX.*

Pada penelitian [14] bertujuan untuk mengetahui performansi masing-masing *controller* melalui parameter *troughput* dan *latency*. Topologi yang digunakan yaitu topologi *single* yang terdiri dari 5 *host*, untuk skenario yang ditetapkan yaitu dengan melakukan 5 kali pengujian di mana pada setiap pengujian jumlah *switch* berubah yaitu 20, 90, 120, 330 dan 450 *switch*. Hasil dari pengujian performa *controller* ini untuk parameter *latency* dan *troughput* pada kedua *controller* cukup baik ketika jumlah *switch* kurang dari 60. Pada *FloodLight*, nilai *latency* tertinggi yaitu 2.780,04 respon/detik yang terjadi saat *switch* berjumlah 20 sedangkan untuk *controller ONOS* memiliki *latency* tertinggi ketika berada pada 30 *switch* dengan *respons* yang diberikan sebesar 5.931 respon/detik. Nilai *troughput* yang dihasilkan oleh *controller FloodLight* sebesar 1.500,38 flow/detik saat memiliki *host* sebanyak 450 sedangkan untuk *ONOS,* nilai *troughput* yang diberikan saat jumlah *host* sebanyak 450 yaitu 354,05 flow/detik.

2.2 DASAR TEORI

2.2.1 *Software Defined Network*

Software Defined Network (SDN) adalah arsitektur jaringan komputer dengan karakteristik dinamis, *manageable, consteffective* serta *adaptable*, hal tersebut yang mendasari *Software Defined Network (SDN)* menjadi arsitektur baru pada bidang jaringan komputer. Karakteristik yang dimiliki oleh arsitektur *Software Defined Network (SDN),* menjadi salah satu dari sekian banyak hal yang dibutuhkan dalam memenuhi aplikasi saat ini yang bersifat dinamis serta memiliki kebutuhan *bandwidth* tinggi. Pada arsitektur *Software Defined Network* terdiri dari tiga layer, yaitu *Application plane, Controller plane dan Data Plane* [15]. *Software Defined Network (SDN)* menjadi sebuah paradigma baru untuk mengontrol serta memanajemen jaringan komputer. *SDN* memisahkan *control plane* dengan *data plane*, dengan adanya pemisahan tersebut seorang administrator dapat memodifikasi protokol di sepanjang perangkat jaringan [16].



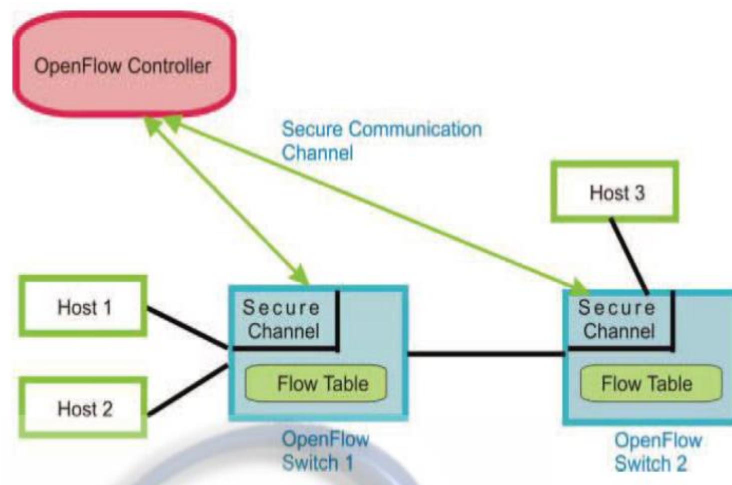
Gambar 2.1 Arsitektur *Software Defined Network* [8].

Pada arsitektur *Software Defined Network* yang terdapat pada Gambar 2.1 terdiri dari komponen, yaitu infrastuktur, kontrol dan aplikasi. Pada infrastruktur terdapat elemen elemen jaringan yang digunakan untuk mengatur *SDN data patch*. Terdapat komponen berupa kontrol yang didalamnya terdapat *controller* untuk mentranslasikan kebutuhan aplikasi dan infrastruktur dengan cara memberikan intruksi-intruksi yang disesuaikan dengan *SDN Datapath* dan *controller* ini dapat memberikan informasi yang relevan. Aplikasi berada pada layer ketiga, sesuai

dengan namanya, layer ini berisi aplikasi aplikasi yang digunakan pada infrastruktur *Software Defined Network* yang berkomunisi dengan sistem melalui *North Bound Interfaces (NBI)* [8].

2.2.2 *Open Flow*

Openflow pada Gambar 2.2 adalah protokol yang terdapat pada arsitektur *Software Defined Network* yang bertugas sebagai jalur komunikasi *control layer* dengan *data layer*. *Open flow* bertugas memeberi akses serta mengatur *forwarding plane* yang terdapat pada *switch* dan *router*. Dilihat dari tugas yang dimiliki oleh *openflow*, ketika paket yang dikirimkan pada *switch openflow* tidak memiliki entri *flow* yang sesuai maka *switch* tersebut akan meneruskan paket menuju *controller*.



Gambar 2.2. Arsitektur *Protocol OpenFlow*

Controller akan melakukan *drop* pada paket paket yang tidak sesuai dengan *flow* lalu *controller* akan menambahkan *entri flow* pada *table flow* untuk paket serta mengintruksikan *switch* untuk meneruskan paket tersebut dengan *entri flow* yang baru. Hal tersebut yang mendasari *open flow* bertugas menjadi jembatan komunikasi yang diatur melalui *controller* yang akan memberikan respon terkait setiap paket yang akan datang [17].

2.2.3 *Mininet*

Pada Gambar 2.3 menujukan logo dari *mininet*, dimana *mininet* sendiri merupakan jaringan *openflow*, mininet menjadi simulator *open source* yang

pertama. *Mininet* juga menjadi *tools* simulator yang sering digunakan pada jaringan *openflow*. *Mininet* mengemas jaringan menjadi sebuah *viritual machine* yang dapat diunduh, dijalankan, diperikasa serta dimodifikasi oleh semua kalangan.

Gambar 2.3. Logo Mininet

Keuntungan yang diberikan oleh *mininet* yaitu baik untuk *prototyping* yang mudah untuk dibagiakan dan mudah untuk digunakan [18]. *Mininet* menjadi sebuah emulator yang menggunakan pendekatan secara *virtual* pada *host, router, switch* dan *link.* Sebagai salah satu emulator jaringan yang realistis, *mininet* dapat membuat topologi jaringan yang cukup kompleks dengan sumber daya yang sedikit dengan menerapkan virtualisasi untuk menjalankan banyak *host* dan *switch* namun hanya menggunakan satu *kernel OS*. Pada penggunaannya, *mininet* ini memanfaatkan *virtual software switch OpenSwitch* yang digunakan dalam membangun berbagai macam jenis topologi pada *OpenFlow Software Defined Network* [17].

2.2.4 *Controller*

Ide terbaru pada arsitektur *Software Defined Network* adalah kemampuan dalam program, memisahkan antara *control plane* dan *data plane*, tidak kalah pentingnya ketika mengelola status jaringan sementara dengan menggunakan kontrol terpusat. Pada sebuah jaringan, *controller* ini merupakan wujud dari kerangka kerja *Software Defined Network* yang diidealkan. Secara teori, *controller* menjadi salah satu elemen vital yang mewujudkan bidang kontrol terdistribusi serta dapat membantu dalam pengawasan pada sebuah arsitektur jaringan. Berbagai *vendor* komersial dan industri pengembangan saling berkompetisi dalam mengembangkan performa *controller*nya. Terdapat banyak *controller* yang tersedia pada *Software Defined Network*, beberapa *controller* bersifat *open source* namun ada beberapa *controller* lain adalah hak milik oleh *vendor*. Contoh *controller* yang bersifat *open source* adalah *OpenDayLight, ONOS, Project Calico, Beacon,*

NOX/POX, Project Floodlight (lisensi beacon dan *apache)*, *OpenVSwitch* (*OVS)*, *Ryu* (oleh lab *NTT*), *Faucet* (*Ryu* berbasis *python* untuk jaringan produksi), *OpenContrail* [19].

2.2.5 POX Controller

Gambar 2.4 Menunjukan logo dari *POX controller* yang masuk dalam kategori *open source* yang menggunakan Bahasa *python*. *POX* secara resmi dapat digunakan pada *Windows, Mac OS* serta *Linux*. *POX* ini beroperasi *sebagai switch, router, load balancing,* perangkat *firewall* dan masih banyak lainnya.

Gambar 2.4. Logo *POX Controller*

POX juga menyediakan arsitektur yang baik dalam arsitektur *Software Defined Network*, beberapa kelebihan yang ada pada *controller POX* :

a. Dapat Menyediakan antarmuka *Pythonic OpenFlow*.
b. Memiliki komponen sampel yang dapat digunakan kembali dalam pemilihan jalur, penemuan topologi, dll.
c. *Controller POX* dapat digunakan pada sistemm operasi apapun yang telah di*install* simulator mininet.
d. Mendukung *Graphical User Interface (GUI)* yang sama dan arsitektur virtual yang serupa dengan *NOX*.
e. *Controller POX* lebih baik dibandingkan dengan *NOX* versi sebelum *POX*.

Script pada *controller POX* dapat diunduh dengan mudah pada Gudang *POX* yang terletak *Github, script* pada *Command Line Interface* (*CLI)* serta *script* lain yang digunakan untuk *controller POX* dapat ditemukan didalamnya [20].

2.2.6 *Ryu Controller*

Ryu adalah salah satu jenis *controller* yang berada pada arsitektur *Software Defined Network*. Pada Gambar 2.5 Menunjukan logo dari *controller RYU*, secara umum fungsi *controller* sama untuk *software defined network* yaitu meningkatkan kemampuan jaringan serta mengkontrol jaringan. *Ryu* termasuk dalam *controller* dalam kategori *open source* yang telah dikembangkan *NTT*.

Gambar 2.5. Logo *RYU Controller*

Pada *RYU Application Program Interface (API)* dapat dikembangkan dengan mudah, *Ryu* menggunakan bahasa *python* sengga mudah dalam penggunakaanya. *Controller Ryu* ini memiliki banyak dokumentasi, dokumentasi inilah yang digunakan peneliti dalam mengembangkan dan menjadikan referensi dalam memecahkan masalah dalam penerlitian. *Controller Ryu* mendukung beberapa *protocol* pada arsitektur *Software Defined Network* seperti *OpenFlow, Netconf, OF- config* dan lainnya [21].

2.2.7 *ONOS Controller*

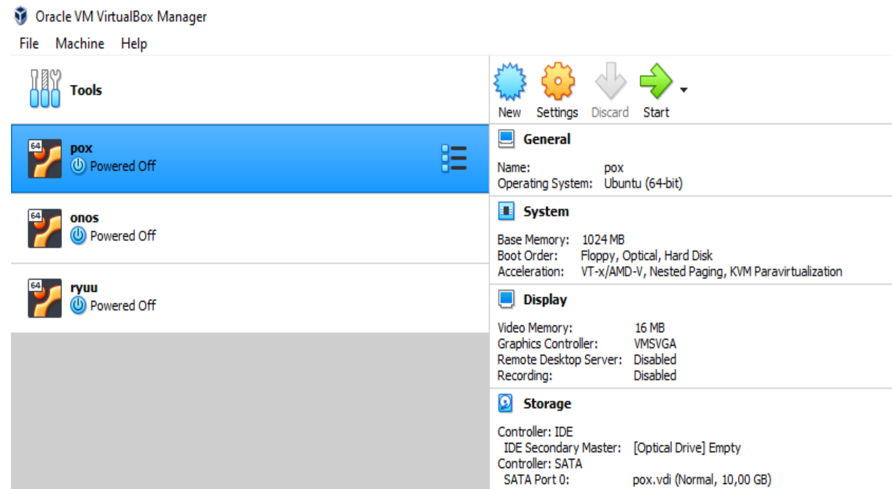
Open Network Operating System (ONOS) yang terdapat pada Gambar 2.6 adalah salah satu dari sekian banyak *controller* yang diimplementasikan pada arsitektur *Software Defined Network, ONOS* termasuk dalam *controller open source* yang berorientasi dengan jaringan operator.

Gambar 2.6. Logo *ONOS Controller*

ONOS menggunakan Bahasa pemograman *Java* menggunakan *OSGi* dalam memanajemen fungsionalitas serta pada setiap fitur yang terdapat dalama *ONOS* diaktifkan dengan menggunakan *Apache*. *ONOS* diliris pada tahun 2014 yang diprakarsai dan dibuat oleh *On. Lab*, *ONOS* menjadi salah satu *controller* yang banyak digunakan pada arsitektur *software defined network* [22].

2.2.8 VIRTUAL MACHINE

Virtual Machine (VM) merupakan salah satu *software* yang digunakan dalam menjalankan program serta menerapkan aplikasi. Pada Gambar 2.7 Merupakan tampilan *software virtual machine* Sehingga dapat menerapkan beberapa sistem operasi dalam satu komputer, setiap sistem operasi menjalankan serta terpisah dengan vm lainnya. *Software virtual machine* ini banyak digunakan dalam menerapkan *cloud,* di era ini layanan *cloud public* mengunakan *virtual machine* hal ini karena menyediakan sumber daya aplikasi *virtual* pada banyak *user* dalam bidang komputasi *virtual machine* ini hemat biaya dan lebih fleksibel. *Virtual Machine (VM)* menjalankan sistem operasi sama seperti sistem operasi utama namun benar-benar terpisah dari jendela aplikasi di *desktop*.

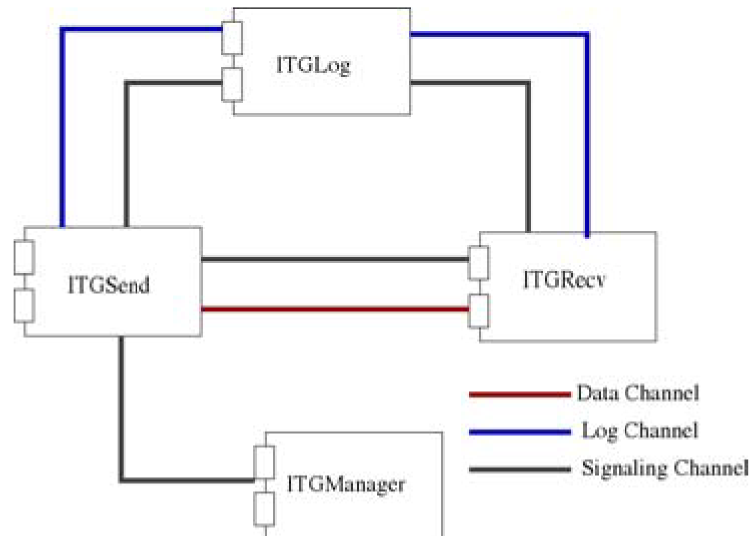


Gambar 2.7. Tampilan Aplikasi *Virtual Machine*

Software VM ini mendukung dalam mengakomodasi kebutuhan dalam menjalankan perangkat lunak yang memerlukan sistem operasi yang berbeda maupun menguji aplikasi pada lingkungan *sandbox* yang aman. *Virtual Machine* telah digunakan pada *virtualisasi server* yang memungkinkan dalam mengkonsolidasikan sumber daya komputasi, *virtual machine* ini juga dapat melakukan tugas yang dianggap beresiko seperti mengakses data yang teinfeksi virus. Karena *virtual machine* ini terpisah dengan sistem utama, maka bagaimanapun kondisi sistem operasi yang terletak pada *virtual machine* tidak akan merusak sistem operasi utama [23].

2.2.9 *D-ITG*

Distributed Internet Traffic Generator (D-ITG) merupakan sebuah platform yang dapat menghasilkan *traffic* dengan menganut pada platform dengan akurat karena *pola traffic* tersebut sudah ditentukan oleh waktu keberangkatan pada paket atau *Inter Departure Time (IDT)* serta pada proses stokastik *Packet Size (PS).*



Gambar 2.8. Arsitektur *D-ITG*

Pada Gambar 2.8 menunjukkan sebuah arsitektur yang dimiliki oleh *D-ITG,* pada gambar tersebut terlihat hubungan antara platform yang disediakan oleh *D-ITG.* Berikut merupakan penjabaran tiga komponen utama yang terdapat pada arsitektur *D-ITG* :

a. ITG Send

ITG send merupakan sebuah komponen pengiriman pada *D-ITG*. Pada *ITG Send* dapat beroperasi pada tiga mode yang berbeda yaitu :

1. Mode aliran tunggal, *ITG send* hanya menghasilkan satu aliran saja. Satu utas tersebut bertanggung jawab dalam membangkitkan arus serta memanajemen dari saluran sinyal melalui *protokol TSP*.
2. Mode beberapa aliran, *ITG send* menghasilkan satu set arus, pada mode ini beroperasi sebagai aplikasi dengan satu utas yang menerapkan prokol *TSP* mendorong proses pembuatan dan utas lain menghasilkan *traffic*
3. Mode daemon, pada mode ini *ITG Send* dikelola melalui jarak jauh oleh *ITGManager* menggunakan *ITGApi*. Dalam mengumpulkan data yang dihasilkan pada saat proses pembangkitan *ITGSend* baik secara lokal maupun jarak jauh menggunakan *server ITGLog*.

b. ITG Recv

Pada *ITGRecv* bekerja sebagai daemon, saat permintaan koneksi *TSP* datang, *ITGRecv* bertanggung jawab dalam mengelola komunikasi

dengan pengirim. *ITGRecv* dapat menyimpan informasi baik secara lokal maupun dalam jarak jauh dengan menggunakan *ITGLog server* log.

c. ITGLog

ITG Log merupakan *server log* yang bekerja pada *server* yang berbeda dengan *host* yang dimiliki *ITGSend* dan *ITGRecv, ITGLog* menyimpan serta menerima informasi *log* dari pengirim dan penerima. Hal tersebut ditangani oleh *signaling protocol, protocol* ini memungkinkan setiap pengirim dan penerima untuk mendaftar dan meninggalkan *server log.* Informasi log dapat dikirim dengan *TCP* dan *UDP* [24].