PEMERATAAN BEBAN JARINGAN MENGGUNAKAN METODE HYBRID (*PROACTIVE* DAN *REACTIVE*) PADA JARINGAN SDN (*SOFTWARE DEFINED NETWORKING*)

TESIS

PROGRAM MAGISTER TEKNIK ELEKTRO
MINAT SISTEM KOMUNIKASI DAN INFORMATIKA

Diajukan untuk memenuhi persyaratan
Memperoleh gelar Magister Teknik

ALHMAD MUKHTAROM

NIM 146060300111004

UNIVERSITAS BRAWIJAYA

FAKULTAS TEKNIK

MALANG

2018

TESIS

PEMERATAAN BEBAN JARINGAN MENGGUNAKAN METODE HYBRID (*PROACTIVE* DAN *REACTIVE*) PADA JARINGAN SDN (*SOFTWARE DEFINED NETWORKING*)

Akhmad Mukhtarom

NIM. 146060300111004

telah dipertahankan didepan penguji
Pada tanggal ……..
dinyatakan telah memenuhi syarat
untuk memperoleh gelar Magister Teknik

Komisi Pembimbing,

Pembimbing I,	Pembimbing II,
Achmad Basuki, S.T., M.MG., Ph.D.	Dr. Ir. Muhammad Aswin, M.T.
NIP. 19741118 200312 1 002	NIP. 19640626 199002 1 001

Malang, Januari 2018

Universitas Brawijaya
Fakultas Teknik, Jurusan Teknik Elektro
Ketua Program Magister Teknik Elektro

Dr. Eng. Panca Mudjirahardjo, ST., MT.
NIP. 19700329 200012 1 001

IDENTITAS TIM PENGUJI TESIS

JUDUL TESIS : Pemerataan Beban Jaringan Menggunakan Metode Hybrid (Proactive dan Reactive) Pada Jaringan SDN (Software Defined Networking)

Nama Mahasiswa : Akhmad Mukhtarom

NIM : 146060300111004

Program Studi : Magister Teknik Elektro

Minat : Sistem Komunikasi dan Informatika

KOMISI PEMBIMBING

Ketua : Achmad Basuki, S.T., M.MG., Ph.D

Anggota : Dr. Ir. Muhammad Aswin, M.T.

TIM DOSEN PENGUJI

Dosen Penguji 1 : Herman Tolle, Dr.Eng., S.T., M.T.

Dosen Penguji 2 : Dr-Ing. Onny Setyawati, S.T., M.T., M.Sc

Tanggal Ujian : 10 Januari 2018

SK Penguji :

Karya ini ku persembahkan untuk

Seluruh rekan, sahabat, saudara dan keluarga

Istri dan putra putri serta ayah dan ibu

PERNYATAAN ORISINALITAS PENELITIAN TESIS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya dan berdasarkan hasil penelusuran berbagai karya ilmiah, gagasan dan masalah ilmiah yang diteliti dan diulas di dalam Naskah Tesis ini adalah asli dari pemikiran saya. Tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu Perguruan Tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan daftar pustaka.

Apabila ternyata di dalam naskah Tesis ini dapat dibuktikan terdapat unsur-unsur jiplakan, saya bersedia Tesis dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, pasal 25 ayat 2 dan pasal 70).

Malang, Januari 2018
Mahasiswa,

Akhmad Mukhtarom
NIM. 146060300111004

RIWAYAT HIDUP

Akhmad Mukhtarom, Pasuruan, 6 Juli 1985, anak pertama dari Wana'i Shofianto dan Lilik Khoiriyah, SDN 1 Warungdowo Pasuruanlang dan SLTP Negeri 6 Pasuruan, SMK Telekomunikasi Sandhy Putra Malang lulus tahun 2000. Studi D3 Teknik Telekomunikasi Politeknik Negeri Malang lulus tahun 2003 dan S1 Teknik Telekomunikasi di Universitas Widyagama Malang lulus pada tahun 2009. Pengalaman kerja sebagai *system* administrator pada unit TIK Universitas Brawijaya Malang sampai sekarang. Melanjutkan studi program Magister (S2) di Program Magister Teknik Elektro Jurusan Elektro Fakultas Teknik Universitas Brawijaya pada tahun 2014-2018.

Malang, Januari 2018

Penulis

UCAPAN TERIMA KASIH

Dalam penyelesaian penelitian tesis ini, penulis banyak mendapatkan bantuan dari berbagai pihak. Untuk itu penulis menyampaikan ucapan terima kasih setulusnya kepada:

1. Achmad Basuki, S.T., M.MG., Ph.D. selaku pembimbing utama yang senantiasa memberikan arahan dan garis besar di setiap bimbingan sehingga benar-benar menyalakan semangat penulis dalam penelitian tesis ini.
2. Dr. Ir. Muhammad Aswin, M.T. selaku pembimbing kedua yang selalu aktif memberikan masukan-masukan teknis sehingga esensi penelitian tesis ini benar-benar muncul ke permukaan.
3. Segenap Sivitas Akademika Fakultas Teknik Jurusan Teknik Elektro Universitas Brawijaya Malang.
4. Seluruh rekan kerja di Unit TIK Universitas Brawijaya yang selalu memberikan semangat.
5. Seluruh keluarga tercinta yang selalu memberikan dukungan dan pengertian dalam proses pengerjaan.

Malang, Januari 2018

Penulis

RINGKASAN

Akhmad Mukhtarom, Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya, Januari 2018, *Pemerataan Beban Jaringan Menggunakan Metode Hybrid (Proactive dan Reactive) pada Jaringan* SDN *(Software Defined Networking*, Dosen Pembimbing: **Achmad Basuki** dan **Muhammad Aswin**.

Mendistribusikan trafik pada topologi jaringan *fat-tree* yang sering digunakan di jaringan *data center* sangatlah penting. *Multipath routing* adalah teknik umum yang digunakan untuk menyeimbangkan trafik. Dalam *Software Defined Networking* (SDN), jalur *routing* sepenuhnya dikendalikan oleh *controller* untuk memilih jalur optimal atau beberapa jalur untuk meningkatkan *throughput trafic-flow* dan menurunkan latensi pengiriman data.

Beberapa metode telah dilakukan pada penelitian sebelumnya diantaranya Slavica dengan mode *controller proactive* dan segmentasi *traffic-flow* dengan mempertimbangkan pemisahan antara TCP dan UDP. Selain itu LABERIO juga telah melakukan pemerataan *traffic-flow* dengan menggunakan mode *controller reactive* dan segmentasi *traffic-flow* dengan mempertimbangkan IP tujuan. Kemudian metode LABERIO disempurnakan dengan metode DLPO dengan tidak melakukan perubahan jalur dan menggantinya dengan merubah nilai *priority* pada *flow-table* untuk mengoptimalkan *traffic-flow* UDP.

Akan tetapi ketika terdapat *traffic-flow* UDP dengan beberapa *traffic-flow* TCP dengan IP tujuan yang sama, pada metode yang digunakan pada penelitian sebelumnya tidak dapat mengakomodasi keadaan tersebut sehingga menyebabkan *traffic-flow* tidak merata ke seluruh *path* yang tersedia. Penelitian ini mengimplementasikan mode hibridisasi *controller proactive-reactive* pada SDN untuk menyeimbangkan trafik. SDN *Controller* memonitor utilisasi jalur dan secara reaktif akan menginstal *flow-tables* ke *switch* yang sesuai setiap kali ada lonjakan trafik yang signifikan untuk periode tertentu. SDN *Controller* akan secara proaktif menerapkan *flow-table* ke *switch* yang sesuai. Metode Hybrid melakukan segmentasi *traffic-flow* dengan mempertimbangkan seluruh elemen *flow* (IP sumber dan tujuan, *internet protocol number* dan *port number* sumber dan tujuan).

Dengan melakukan beberapa pengujian, diantaranya adalah dengan memberikan beberapa traffic-flow TCP dan UDP secara bersamaan, hasil evaluasi pada topologi *fat-tree*

menunjukkan bahwa bahwa mode hibridisasi yang diusulkan berkinerja lebih baik daripada LABERIO dengan *throughput traffic-flow* rata – rata sebesar 7,558 Mbps (25,193%) dari Slavica, 2,38 Mbps (7,93%) dari LABERIO dan 3,44 Mbps (11,47%) dari DLPO. Pengujian latensi pengiriman data juga dilakukan dengan memberikan proses copy file dan mendapatkan hasil untuk latensi pengiriman data, metode dapat menurunkan rata – rata sebesar 31 detik dari Slavica, 18 detik dari LABERIO dan 21 detik dari DLPO. Pada kondisi dimana terjadi traffic-spike, metode Hybrid dapat mengakomodasi dengan memberikan time-threshold sebesar 15 detik sehingga tidak melukan perhitungan ulang.

Kata kunci : SDN, *fat-tree*, *data center*, utilisasi *bandwidth*, *multipath*, *load balance*

SUMMARY

Akhmad Mukhtarom, *Department of Electrical Engineering, Faculty of Engineering, Unversity of Brawijaya, January 2018, Detect and Control Network Congestion with Open Shortest Path First, Academic Supervisor:* **Achmad Basuki** and **Muhammad Aswin**.

Distributing traffic on a fat-tree network topology that is often used in data center networks is very important. Multipath routing is a common technique used to balance traffic. In Software Defined Networking (SDN), the routing path is completely controlled by the controller to select the optimal path or multiple paths to increase the traffic-flow throughput and decrease data transmission latency.

Several methods have been done in previous researches including Slavica with proactive controller mode and traffic-flow segmentation by considering the separation between TCP and UDP. In addition LABERIO has also conducted traffic-flow equalization by using reactive controller mode and traffic-flow segmentation by considering the destination IP address. Then LABERIO method is enhanced by DLPO method by not making path changes and replacing it by changing the priority value in the flow-table to optimize UDP traffic-flow.

However, when there is UDP traffic-flow with some TCP traffic-flow with the same destination IP address, the methods used in the previous study can not accommodate the situation, causing traffic-flow not distribute to all available paths. This study implements a proactive-reactive controller hybridization mode in the SDN to balance traffic. The SDN Controller monitors the path utilization and will reactively install flow-tables to the appropriate switches whenever there is significant traffic spikes for a given period. SDN Controller will proactively apply flow-table to the appropriate switch. Hybrid method performs traffic-flow segmentation by considering all flow elements (source and destination IP, internet protocol number and source and destination port number).

By performing several tests, among others, by providing multiple TCP and UDP traffic-flow simultaneously, the results of the evaluation on the fat-tree topology show that the proposed hybridization mode performs better than LABERIO with an average traffic-

flow throughput of 7.558 Mbps (25.193%) of Slavica, 2.38 Mbps (7.93%) of LABERIO and 3.44 Mbps (11.47%) of DLPO. Testing data transmission latency is also done by giving the process of copying files and getting results for data transmission latency, the method can decrease the average of 31 seconds from Slavica, 18 seconds from LABERIO and 21 seconds from DLPO. In conditions where traffic-spike occurs, the Hybrid method can accommodate by giving a time-threshold of 15 seconds so as not to re-calculate.

Keywords : *SDN, fat-tree, data center, bandwidth utilization, multipath, load balance*

KATA PENGANTAR

Dalam tulisan ini, penulis mengacu pada ketentuan pembuatan tesis yang berlaku pada Program Master dan Doktor Fakultas Teknik Universitas Brawijaya Malang. Dalam melengkapi proposal ini, digunakan berbagai macam referensi penunjang yang terkait dengan permasalahan yang akan diteliti. Penulis menyadari bahwa proposal ini dapat diselesaikan karena bantuan dan dukungan dari berbagai pihak terutama kedua pembimbing, untuk itu penulis menghaturkan banyak terimakasih, khususnya kepada :

1. Achmad Basuki, S.T., M.MG., Ph.D., selaku Pembimbing I yang telah banyak meluangkan waktunya membimbing dan mengarahkan penulis.
2. Dr. Ir. Muhammad Aswin, M.T., selaku Pembimbing II yang telah banyak meluangkan waktunya membimbing dan mengarahkan penulis.
3. Rekan-rekan mahasiswa Strata Dua (S2) Program Studi Teknik Elektro Minat Sistem Komunikasi dan Informatika angkatan 2014 Universitas Brawijaya Malang, yang telah memberikan dukungan moril kepada penulis.
4. Rekan-rekan kerja di Unit TIK Universitas Brawijaya Malang, yang telah memberikan semangat dan kesempatan untuk dapat menggunakan fasilitas laboratorium untuk terselesaikannya penelitian tesis ini.

Akhirnya, semoga penelitian tesis ini mendapatkan perhatian dari semua pihak.

Malang, Januari 2018

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR LAMPIRAN

BAB I
PENDAHULUAN

1.1. Latar Belakang

Data center dimana di dalamnya terdapat banyak *server-node* membutuhkan jaringan berkapasitas tinggi antar *server-node* yang ada di dalamnya. Kebanyakan *data center* menggunakan topologi jaringan *fat-tree*. Topologi jaringan *fat-tree* dapat memberikan kapasitas koneksi yang tinggi antar *server-node* (Yao, Wu, Venkataramani, & Subramaniam, 2014). Dengan beberapa *path* yang tersedia dalam topologi *fat-tree* memungkinkan *server-node* satu dengan yang lain dapat menggunakan beberapa *path* untuk menambah performa komunikasi antar *server-node*. Salah satu indikasi performa komunikasi antar *server-node* adalah *throughput* yang diperoleh trafik antar *server-node*. *Multipath* menyediakan beberapa *redundant link* untuk mendukung *availability* sumber daya jaringan yang tinggi. Akan tetapi keberadaan *multipath* dianggap suatu pemborosan sumber daya jaringan ketika *path* yang tersedia tidak digunakan secara optimal. Beban trafik yang tidak merata akan mempengaruhi *throughput* antar *server-node*.

Pada teknologi jaringan konvensional metode pemerataan jaringan dapat menggunakan ECMP (*Equal Cost Multipath*) yang digunakan pada metode *routing* OSPF (*Open Short Path First*). Alternatif lain adalah menggunakan pendekatan teknologi SDN (*Software Defined Networking*). Metode ECMP pada OSPF tidak dapat secara langsung diterapkan pada teknologi jaringan SDN. Hal ini dikarenakan ECMP OSPF bekerja berdasarkan IP *address* tujuan sedangkan SDN bekerja berdasarkan *flow*. Trafik dalam SDN selanjutnya disebut sebagai *traffic-flow*. SDN sebagai paradigma baru dalam dunia jaringan memberikan peluang untuk pengembangan metode pemerataan trafik kepada beberapa *path* pada jaringan. Karena SDN bekerja berdasarkan *flow*, maka untuk meneruskan suatu *traffic-flow* dapat mempertimbangkan IP *address* (sumber dan tujuan), *protocol* dan *port number* (sumber dan tujuan) yang digunakan.

Fenomena *traffic-spike* pada *data center* adalah kondisi dimana terjadi peningkatan *traffic-flow* secara signifikan dalam rentang waktu yang singkat. Fenomena ini juga perlu mendapat perhatian pada metode pemerataan beban *traffic-flow*. Sistem pemerataan yang

hanya memperhatikan parameter nilai beban *load-traffic* yang tinggi akan mengakibatkan melakukan proses pemerataan berulang – ulang ketika terjadi *traffic-spike*.

Pada beberapa penelitian sebelumnya mencoba melakukan pemerataan beban *traffic-flow* kepada beberapa *path* yang ada dalam jaringan SDN. Pengalihan *traffic-flow* ketika salah satu *path* jaringan mencapai 80% kapasitas maksimal (Lazuardi, 2016). Dengan menggunakan *controller* mode *reactive* menyebabkan *controller* memproses setiap *traffic-flow* yang terjadi pada jaringan sehingga *throughput* yang diperoleh tidak optimal. Lazuardi hanya mempertimbangkan IP *address* sumber dan tujuan sehingga tidak dapat memisahkan *traffic-flow* kapasitas besar dengan IP *address* sumber dan tujuan yang sama. Akibatnya *traffic-flow* tersebut tidak dapat menggunakan beberapa *route-path*. *Route-path* adalah deretan *path* yang dilalui sebuah *traffic-flow* dari IP *address* sumber menuju IP *address* tujuan. Parameter penentu untuk menyatakan keadaan *overload-path* hanya berdasar nilai beban *traffic-load* sesaat menjadikan metode Lazuardi sangat sensitif terhadap *traffic-spike*. Pada penelitian lain, Slavica mencoba mensegmentasi *traffic-flow* lebih detail dengan ikut membedakan parameter *internet protocol* TCP dan UDP yang digunakan oleh setiap *traffic-flow* (Slavica Tomovic, 2016). Slavica menggunakan *controller* dengan metode *proactive* sehingga *controller* hanya memproses pada saat awal terjadinya *traffic-flow*. Sehingga beban kerja *controller* tidak seberat pada penelitian Lazuardi. Slavica tidak mempertimbangkan penanganan *traffic-flow* ketika terjadi perubahan *route-path*, sehingga memungkinkan terdapat bagian *traffic-flow* yang *time*out. Sama seperti halnya pada penelitian Lazuardi, Slavica juga hanya menggunakan nilai beban *traffic-load* sesaat sebagai parameter penentu dalam menyatakan *overload-path*. Kedua penelitian tersebut (Lazuardi dan Slavica) menggunakan dasar algoritma *Dijkstra* dalam menentukan rute *traffic-flow*.

Penelitian lain menggunakan tabel *path* sebagai daftar *path* yang dapat dilewati untuk menentukan rute *path* yang akan dilalui oleh sebuah *traffic-flow* (Hui Long, 2013). Hui Long menamakan metode yang digunakan dengan LABERIO (*Dynamic load-balanced Routing in Openflow-enabled*). LABERIO menggunakan topologi *fat-tree*, dimana topologi telah digunakan oleh kebanyakan *data center*. LABERIO membedakan *traffic-flow* hanya berdasarkan IP *address* sumber dan tujuan sama halnya pada penelitian Lazuardi. Mode *controller proactive* yang digunakan LABERIO dan tabel *path* yang digunakan menurunkan *throughput traffic-flow* saat terjadi proses pengalihan *route-path*. Sama seperti halnya pada penelitian Lazuardi, LABERIO juga hanya menggunakan nilai beban *traffic-load* sesaat

sebagai parameter penentu dalam menyatakan *overload-path*. Penelitian Hui Long disempurnakan oleh Yuan Liang dengan menghilangkan proses pengalihan *route-path traffic-flow* dan digantikan dengan proses merubah nilai prioritas dari *traffic-flow* (Yuan-Liang Lan, 2016). Yuan Liang menamakan metode yang diterapkan dengan nama DLPO (*Dynamic load-balanced path optimization in SDN-based data center networks*). Akan tetapi DLPO juga belum dapat mengatasi masalah *traffic-spike* yang terjadi.

Pada penelitian dahulu yang telah dibahas hanya menggunakan salah satu mode *controller* yaitu *reactive* atau *proactive*. *Controller reactive* harus bekerja secara ekstra untuk memproses setiap *traffic-flow* yang terjadi pada jaringan. Metode *proactive* tidak dapat beradaptasi dengan perubahan beban *traffic-flow* yang terjadi sehingga lebih sulit dalam melakukan pengalihan *traffic-flow*. Segmentasi *traffic-flow* yang detail dan berkapasitas kecil akan mempermudah dalam pemerataan pada *path* jaringan. Akan tetapi semakin detail segmentasi akan semakin membebani *controller* pada saat proses perhitungan, maka diperlukan pemisahan antara *controller* sebagai *control-plane* dan proses perhitungan sebagai *application-plane*. Untuk mengatasi beberapa masalah tersebut maka dalam penelitian ini digunakan *controller* dengan mode *proactive* dan *reactive*. *Controller proactive* dalam menentukan *route-path traffic-flow* berdasarkan data *monitoring* dan *reactive* dalam memproses *traffic-flow* ketika terjadi *overload-path* dan melakukan perhitungan ulang sesuai data *monitoring*. Selain mode *controller*, dalam penelitian ini juga menggunakan segmentasi *traffic-flow* yang detail dengan mempertimbangkan IP *address* (asal dan tujuan), *protocol* dan *port number* (asal dan tujuan). Dengan segmentasi yang detail, agar tidak membebani kinerja *controller*, maka dalam penelitian ini dilakukan pemisahan antara *control-plane* dan *application-plane*. Dalam penelitian ini digunakan dua parameter dalam memutuskan kapan proses perhitungan ulang *route-path* dilakukan. Kedua parameter tersebut adalah *load-threshold* dan *time-threshold*. *Load-threshold* adalah batas ambang nilai beban *traffic-load* total pada sebuah *path* untuk dinyatakan sebagai *overload-path*. *Time-threshold* adalah batas ambang durasi terjadinya *overload-path* untuk dinyatakan perlu dilakukan proses perhitungan ulang *route-path* semua *traffic-load* yang terjadi. *Time-threshold* sangat berguna untuk menghadapi *traffic-spike* agar tidak terjadi proses perhitungan yang berulang – ulang. Dengan demikian diharapkan penelitian ini dapat lebih meratakan beban *traffic-flow* kepada seluruh *path* yang ada pada jaringan sehingga perolehan *throughput* setiap *traffic-flow* lebih optimal.

1.1. Identifikasi Masalah

Berdasarkan latar belakang maka dapat diidentifikasi jika terdapat *traffic-flow* dengan *internet protocol* dan tujuan IP yang sama akan mengakibatkan beban *traffic-flow* pada *path* jaringan tidak merata. Metode pemerataan yang dilakukan oleh Slavica, LABERIO dan DLPO tidak dapat maengakomodasi keadaan tersebut sehingga *traffic-flow* akan dilewatkan dengan rute *path* yang sama sehingga mengakibatkan penyebaran *traffic-flow* tidak merata pada seluruh *path*. Tidak meratanya beban *traffic-flow* pada jaringan *data center* dengan topologi *fat-tree* dengan menggunakan teknologi SDN menyebabkan perolehan *throughput traffic-flow* tidak optimal. Perolehan *throughput* yang rendah menambah waktu pertukaran data antar *server-node* sehingga mempengaruhi performa *data center* secara keseluruhan.

1.2. Rumusan Masalah

Dari uraian latar belakang dan identifikasi masalah di atas, maka fokus permasalahan pada penelitian ini adalah sebagai berikut :

a. Bagaimana penerapan mode *controller* secara *reactive* dan *proactive* (Hybrid) dalam melakukan pemerataan beban *traffic-flow* guna meningkatkan optimalisasi penggunaan *path* pada jaringan.
b. Bagaimana pengaruh segmentasi *traffic-flow* dalam proses pemerataan beban *traffic-flow* kepada seluruh *path* pada jaringan.
c. Apa pengaruh terhadap perolehan nilai *throughput* dari *traffic-flow* secara keseluruhan.

1.3. Lingkup Kajian

Adapun lingkup kajian dalam penelitian ini adalah sebagai berikut :

1. Pada penelitian ini tidak membahas optimasi metode pembacaan data statistic jaringan. Pembacaan data statistic jaringan hanya menggunakan aplikasi pembantu untuk sistem *monitoring sFlow-RT* dan *bwm-ng*.
2. Arsitektur jaringan menggunakan arsitektur *Software defined networking* (SDN).
3. *Controller* SDN yang digunakan adalah *Ryu*.
4. *Switch* yang digunakan adalah *open-vSwitch*.
5. *Protocol* yang digunakan pada *open-vSwitch* adalah *openflow*.

6. *Environment* simulasi yang digunakan adalah *mininet*.
7. Segmentasi paket TCP/IP menggunakan kombinasi IP *address* (sumber dan tujuan), *protocol* TCP/IP dan *port number*.
8. Algoritma yang digunakan adalah *Dijkstra* dengan perubahan nilai *path-cost* yang berubah secara dinamis sesuai dengan nilai yang diberikan pada setiap iterasi segmen *traffic-flow*.

1.4. Tujuan Penelitian

Tujuan dalam penelitian ini adalah untuk mengatasi kelemahan sistem pemerataan *traffic-flow* pada teknologi jaringan SDN dengan topologi *fat-tree* pada *data* agar dapat meningkatkan *throughput traffic-flow* dan menurunkan latensi pengiriman data. Metode pemerataan dengan menggunakan hibridasi mode proactive dan reactive controller dan segmentasi *traffic-flow* yang mempertimbangkan

1.5. Manfaat Penelitian

Dengan penelitian ini diharapkan memberikan kontribusi terhadap sistem pemerataan beban *traffic-flow* pada *path* jaringan *data center* agar dapat meningkatkan *throughput traffic-flow* dan menurunkan latensi pengiriman data.

BAB II
DASAR TEORI

2.1. Penelitian Terdahulu

Penelitian SDN menggunakan algoritma *Dijkstra* sudah banyak dilakukan. Beberapa penelitian juga telah mengembangkan kinerja permodelan ini. Pada dasarnya *Dijkstra* mempunyai parameter *weight/cost* pada setiap *path*-nya. *Dijkstra* tidak melihat perbedaan perangkat (*node*) dan menganggap semua perangkat (*node*) mempunyai performa yang sama. *Extended-Dijkstra* telah menambahkan nilai *weight/cost* untuk setiap perangkat. *Extended-Dijkstra* dapat mengakomodir pengaruh perbedaan perangkat ke dalam perhitungannya (Jehn-Ruey Jiang, 2014). Akan tetapi pada penelitian ini masih belum memperhitungkan kondisi beban *traffic-flow* pada setiap *path*. Nilai *weight* pada *path* dan *node* adalah sebuah nilai statis yang ditentukan di awal, sehingga memungkinkan akan terjadinya suatu *path* dengan beban *traffic-flow* yang berlebih (*overload-path*).

Untuk dapat melihat beban *traffic-flow* pada setiap *path* digunakan aplikasi (*software*) *sFlow-RT* yang berkomunikasi secara asynchronous dengan *controller*. Ketika suatu *path* dengan beban *traffic-load* melebihi dari 80% kapasitas maksimal *path* tersebut, maka proses perhitungan algoritma *Dijkstra* akan dilakukan untuk memisahkan dan memberikan rute baru bagi beberapa *traffic-flow* yang melewati *overload-path* tersebut (Lazuardi, 2016). Segmentasi *traffic-flow* dibedakan berdasar IP *address*. Jika terdapat *traffic-flow* dengan IP *address* sumber dan tujuan yang sama dengan kapasitas melebihi 80% nilai kapasitas maksimum *path*, maka akan menimbulkan proses perhitungan secara berulang – ulang (*isolation-process*). Hal tersebut dikarenakan pada penelitian ini *traffic-flow* dengan IP *address* yang sama akan dinyatakan sebagai segmen *traffic-flow* yang sama. Segmen *traffic-flow* yang sama akan melewati rute *path* yang sama. Akan tetapi penelitian ini sudah mengimplementasikan sistem *monitoring* beban *traffic-flow* pada seluruh *path* jaringan secara *real-time*.

Metode lain untuk menyamakan beban *traffic-load* antar *path* pada jaringan adalah dengan mengurangi tabel *flow* pada *node* pengirim pada *path* tersebut dan dialihkan ke *node* yang lain agar *traffic-flow* beralih pada *path* yang lain (Yi-Chih Lei, 2015). Akan tetapi jika diterapkan pada kondisi yang telah dijelaskan pada pendahuluan, metode pada penelitian ini

tidak sepenuhnya dapat mengakomodir kondisi tersebut. Dikarenakan pada metode ini segmentasi *traffic-flow* hanya berdasarkan IP *address*, maka segmen trafik masih terlalu besar dan pemerataan beban *traffic-flow* antar *path* akan lebih sulit dilakukan.

Pada penelitian lain memberikan pernyataan bahwa pada jaringan yang digunakan *traffic-flow* UDP jauh lebih besar dari pada *traffic-flow* TCP, maka dianggap perlu pemisahan antara jalur *traffic-flow* UDP dan jalur untuk *traffic-flow* TCP. Dengan demikian *traffic-flow* TCP tidak perlu berdesakan pada *path* yang sama dengan *traffic-flow* UDP. Dan ini meningkatkan performa *traffic-flow* yang menggunakan protokol TCP (Slavica Tomovic, 2016). Segmentasi *traffic-flow* pada penelitian yang dilakukan Slavica sudah menggunakan *layer* 4 *OSI* yaitu membedakan *traffic-flow* TCP dan UPD, akan tetapi belum menggunakan parameter *port number*, sehingga ukuran segmen *traffic-flow* masih besar. Hal ini dikarenakan pada statistik jaringan yang digunakan hanya membutuhkan solusi pemisahan beban *traffic-flow* TCP dan UDP. Untuk keadaan yang telah diceritakan pada pendahuluan, maka segmentasi *traffic-flow* pada penelitian ini belum dapat memberikan solusi yang tepat.

Ketika suatu jaringan dipenuhi *traffic-flow* TCP dan UDP yang sama – sama besar, sedangkan ada beberapa aplikasi yang sifatnya memerlukan respon komunikasi yang cepat dibanding aplikasi yang lainnya. Dalam kondisi seperti ini diperlukan sistem prioritas untuk melewatkan *traffic-flow* aplikasi tersebut. Setiap aplikasi mempunyai protokol dan *port number* khusus untuk berkomunikasi. Segmentasi yang lebih detail dibutuhkan pada kondisi seperti ini. Mengingat acuan yang digunakan adalah *service* aplikasi yang berjalan dengan protokol dan *port number* tertentu, maka segmentasi dilakukan dengan melihat dua parameter tersebut (Shih-Chun Lin, 2016). Meskipun segmentasi dilihat dari protokol dan *port number*, secara prinsip tidak akan terlepas dari IP *address*. Dalam penelitian ini segmentasi *traffic-flow* sudah menjadi segmen yang kecil. Penelitian ini menggunakan metode *inforcement learning* dan *Q-learning*, dimana metode ini berdasarkan data balikan terhadap suatu aksi dari sistem. Metode ini cocok untuk suatu jaringan dimana *controller* tidak dapat mengetahui secara pasti keadaan *router* dan hanya menilai dari balikan suatu aksi dari sistem. Dengan sistem *monitoring* menggunakan *sFlow-RT* hal tersebut tidak perlu dilakukan, karena *sFlow-RT* sudah mengirimkan informasi yang detail tentang keadaan *switch*/*router*. Tentunya hal ini akan lebih efisien dan lebih valid dari sekedar membaca nilai balikan dan mempelajarinya.

Penelitian lain melakukan pemerataan beban *traffic-flow* pada jaringan *data center* dengan topologi jaringan *fat-tree*. Semua kemungkinan *path* yang akan dilalui dimuat dalam tabel dan dijadikan data acuan untuk menentukan rute *path* yang akan dilalui oleh *traffic-flow*. Metode ini tidak menggunakan *Dijkstra* melainkan menggunakan tabel yang dideklarasikan pada awal sistem berjalan (Hui Long, 2013). Kemudian disempurnakan oleh Yuan-Liang dengan menghilangkan proses pemindahan rute dan mengganti dengan proses mengurangi *flow-table priority* untuk packet-*flow* terbesar yang membebani *overload-path* (Yuan-Liang Lan, 2016). Pada kedua penelitian tersebut sudah menggunakan topologi yang digunakan oleh kebanyakan *data center*, akan tetapi segmentasi masih berdasar IP *address*. Segmentasi *traffic-flow* belum detail dan belum dapat mengakomodir keadaan yang telah dijelaskan pada bab pendahuluan.

Ringkasan penelitian terdahulu dalam bentuk tabular ditampilkan pada tabel 2.1.

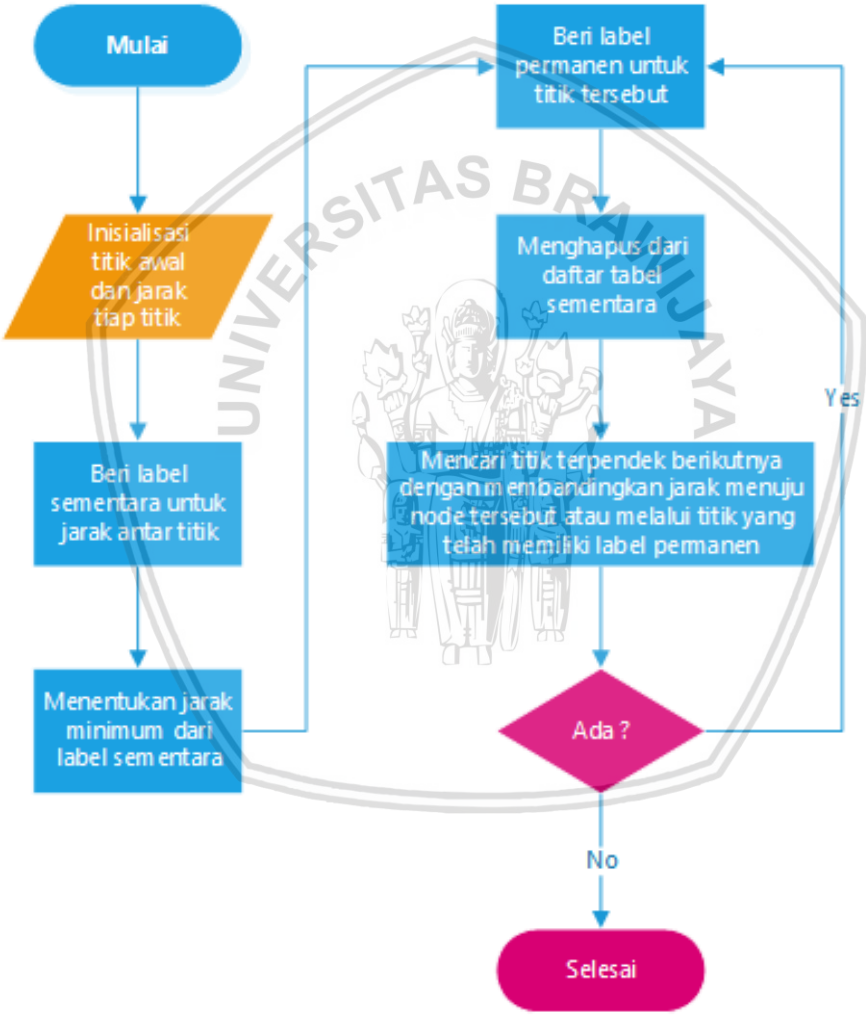
Tabel 2.1. Ringkasan penelitian terdahulu

Penelitian	***Dynamic Routing***	***Traffic Balancing***	***Segmentation***			***Monitoring***	
			IP	***Protocol***	***Port***	***Learning***	***Counting***
Extending Dijkstra's Shortest Path Algorithm for Software Defined Networking (Jehn-Ruey Jiang, 2014)	☑	☒	☑	☒	☒	☒	☒
Metode Pemilihan Jalur *Routing* Adaptif Berdasar Kemacetan Jaringan Dengan Algoritma *Dijkstra* Pada *Openflow Network* (Lazuardi, 2016)	☑	☑	☑	☒	☒	☒	☑
Multipath Routing in SDN-*based Data center Networks* (Yi-Chih Lei, 2015)	☑	☑	☑	☒	☒	☒	☑
A new approach to dynamic routing in SDN *networks* (Slavica Tomovic, 2016)	☑	☑	☑	☑	☒	☒	☒
*Dynamic load-balanced routing in Openflow-enabled network*s (Hui Long, 2013)	☑	☑	☑	☑	☒	☒	☑
Dynamic Load-balanced Path Optimization in SDN-*based Data center Networks* (Yuan-Liang Lan, 2016)	☑	☑	☑	☑	☒	☒	☑

2.2. Dijkstra

Algoritma Dijkstra adalah sebuah algoritma yang sangat efisien dan banyak digunakan untuk menentukan jalur terpendek dari node satu ke node yang lain dalam sebuah data graph. Data graph adalah permodelan data yang digunakan dalam algoritma Dijkstra. Data graph terdiri dari beberapa node dan garis penghubung berarah antar node. Setiap garis hubung diberikan nilai yang disebut cost atau weight. Jarak terpendek adalah nilai total cost dari garis penghubung yang dilewati dari node sumber ke node tujuan (Fitria, 2013).

Berikut adalah *flowchart* dari algoritma *Dijkstra* :



Gambar 2.1. Flowchart algoritma Dijkstra

Sumber : Simulasi Algoritma Dijkstra Pada Protokol Routing Open Shortest Path First (Suherman, 2011)

Gambar 2.1 adalah *flowchart* dasar dari algoritma *Dijkstra*, dimana *Dijkstra* akan memberikan label kepada titik atau *node* dan memberikan nilai terhadap jarak antar *node*.

Nilai ini disebut juga sebagai *cost/weight*. *Dijkstra* akan menghitung berbagai kemungkinan jalur yang ditempuh dan akan mengambil jalur dengan total *cost* terendah sebagai hasilnya.

Algoritma *Dijkstra* dalam bahasa pemrograman *python* dapat menggunakan *library networkx* (NetworkX, 2017). Untuk menambahkan *library networkx* pada *environment python* data menggunakan perintah :

```
root@tiram-devel:~# sudo pip install networkx
```

Langkah – langkah untuk menggunakan modul *networkx* dalam pemrograman *python* adalah sebagai berikut:

- *Import* modul *networkx* pada *script python*.

  Berikut sintaks dalam bahasa pemrograman *python* :

```
import networkx as nx
from networkx import *
from networkx.readwrite import json_graph
```

- Membuat permodelan data *graph*.

  Data *graph* membutuhkan 2 jenis data yaitu *node* dan *edge*. *Node* adalah titik yang diumpamakan sebagai *host* atau *switch*, sedangkan *edge* adalah koneksi atau *link* antar *node*. Berikut sintaks dalam bahasa pemrograman *python*:

```
G.add_node(1)
G.add_nodes_from([2, 3])
G.add_edge(1, 2)
e = (2, 3)
G.add_edge(*e)
G.add_edges_from([(1, 2), (1, 3)])
```

- Memberikan nilai *cost* atau *weight* pada *edge*.

  Berikut sintaks dalam bahasa pemrograman *python* :

```
G[1][2]['weight']= 100
```

- Menghitung jarak terpendek antara dua *node*.

  Berikut sintaks dalam bahasa pemrograman *python* :

```
routeNode = nx.Dijkstra_path(G,1,3)
```

2.3. Software Defined *Networking*

Prinsip dasar dalam *Software Defined Network* (SDN) adalah pemisahan antara *control-plane* dan *data-plane*. Dalam SDN komunikasi antara *data-plane* dan *control-plane* diatur dalam suatu protokol yang telah distandarisasi. Peran *controller* atau *control-plane*

adalah sebagai kontrol bagaimana *data-plane* bekerja. *Data-plane* hanya akan melakukan aksi sesuai *rule* yang telah diberikan oleh *controller*.

Beberapa aspek penting dari 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 dan kemudian mempresentasikannya melalui (sejenis) API (*Application Programming Interface*).
4. *Virtual*isasi dimana beberapa sistem operasi jaringan dapat mengkontrol bagian-bagian (*slices* atau *substrates*) dari perangkat yang sama.

Sebelumnya telah disinggung tentang inovasi. Dalam hal ini terkait dengan kebutuhan inovasi untuk bidang jaringan yang semakin kompleks. Termasuk diantaranya adalah fakta-fakta dan kebutuhan berikut:

1. *Virtual*isasi dan *cloud*

   Komponen dan entitas jaringan Hybrid - antara fisik *bare metal* dan yang *virtual*.
2. *Orchestration* dan *scalability*

   Kemampuan untuk mengatur dan mengelola ribuan perangkat melalui sebuah *point of management*
3. *Programmability* dan *automation*

   Kemampuan untuk mengubah *behavior* (perilaku) jaringan serta untuk dapat melakukan perubahan tersebut secara otomatis (sebagai contoh adalah kemampuan *troubleshooting*, perubahan *policy* dan lain-lain)
4. *Visibility*

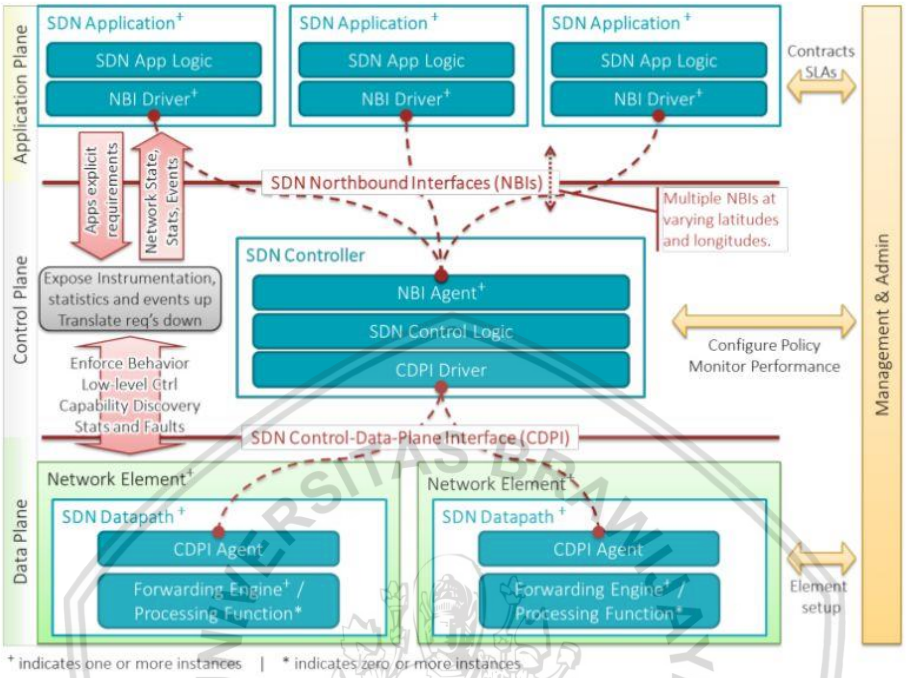
   Kemampuan untuk dapat memonitor jaringan, baik dari sisi sumber daya, konektivitas dan lain-lain.
5. Kinerja

   Kemampuan untuk memaksimalkan penggunaan perangkat jaringan, misalnya optimasi *bandwidth, load balancing, traffic engineering* dan lain-lain (berhubungan dengan *Programmability* dan *Scalability*)

Dalam konsep SDN, tersedia *open interface* yang memungkinkan sebuah entitas *software*/aplikasi untuk mengendalikan konektivitas yang disediakan oleh sejumlah sumber-

daya jaringan, mengendalikan aliran trafik yang melewatinya serta melakukan inspeksi terhadap atau memodifikasi trafik tersebut.

Gambar berikut menunjukkan arsitektur SDN beserta komponen dan interaksinya.



Gambar 2.2. Arsitektur SDN.

Sumber : Arsitektur SDN (SDXCentral, 2012) - https://www.sdxcentral.com/sdn/definitions/inside-sdn-architecture/

Arsitektur SDN dapat dilihat pada gambar 2.2 yang terbagi menjadi 3 bagian:

- *Data plane*

  Berfungsi sebagai *device* atau piranti untuk meneruskan *flow* dengan *rule* yang telah diberikan oleh *controller* (*control plane*). Jika *data plane* tidak dapat memutuskan untuk meneruskan sebuah *flow*, maka *data plane* akan berkomunikasi dengan *controller* untuk melakukan keputusan.

- *Control plane*

  Berfungsi sebagai pemberi keputusan dan penerapan *rule* terhadap *data plane.*

- *Application plane*

  Pada kenyataannya *application plane* ini adalah sebuah *optional modul* yang dapat ditambahkan. *Application plane* dapat digabungkan dengan *control plane* atau terpisah dengan *control plane*. *Application pl*ane berfungsi untuk menjalankan algoritma atau logika dari kinerja jaringan secara keseluruhan. *Application plane*

berkomunikasi dengan *controller* untuk menerapkan hasil algoritma atau logika jaringan pada *data plane*.

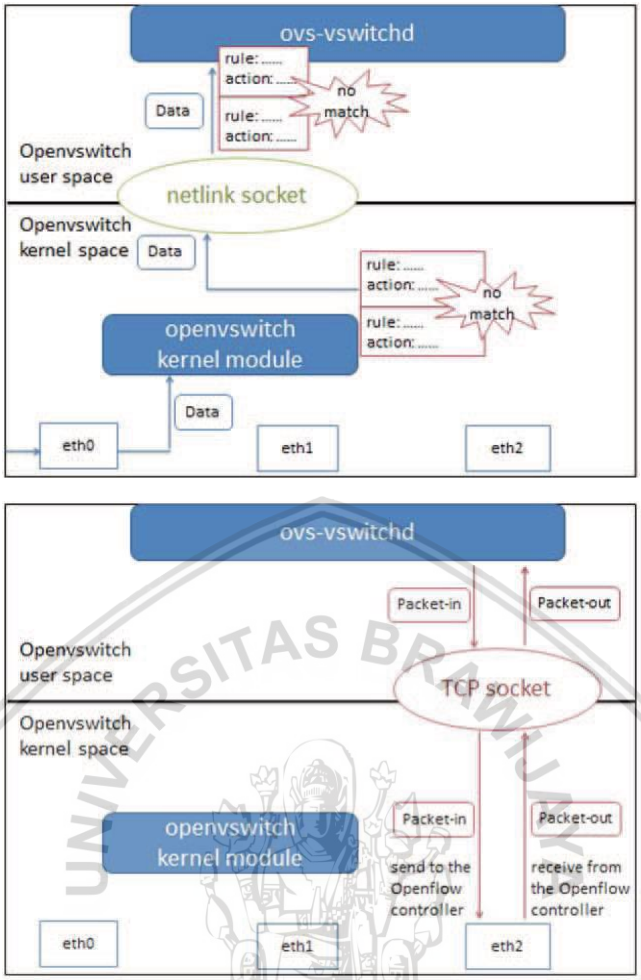
Bidang manajemen dan admin bertanggung jawab dalam inisialisasi elemen jaringan, menghubungkan SDN *Datapath* dengan SDN *Controller*, atau menkonfigurasi cakupan (*coverage*) dari SDN *Controller* dan SDN *Application*.

Arsitektur SDN seperti dijelaskan di atas, dapat berjalan paralel dengan jaringan non-SDN, fitur yang sangat berguna untuk migrasi secara bertahap menuju jaringan SDN.

Konsep *layer* (lapis enkapsulasi) merupakan contoh dan satu-satunya abstraksi jaringan sebelum era SDN. Konsep *layer* ini juga hanya terbatas meng-abstraksi-kan *data-plane*: tidak ada konsep serupa untuk *control-plane*. Setiap kebutuhan baru untuk kontrol jaringan, dilakukan melalui mekanisme (protokol). Tidak ada yang salah dengan hal ini, selama kita mampu mengelola kompleksitas (*mastering complexity*) (Komunitas SDN-RG - Institut Teknologi Bandung, 2014).

2.4. Open-v*Switch*

Openvswitch terdiri dari beberapa modul yaitu *user-space daemon* (*ovs-vswitchd*), *server database* (*ovsdb-server*) dan modul *kernel*. *Flow* yang diterima oleh *openvswitch* hanya akan diproses oleh modul *kernel openvswitch* dan *daemon* (Shie-Yuan, 2015).



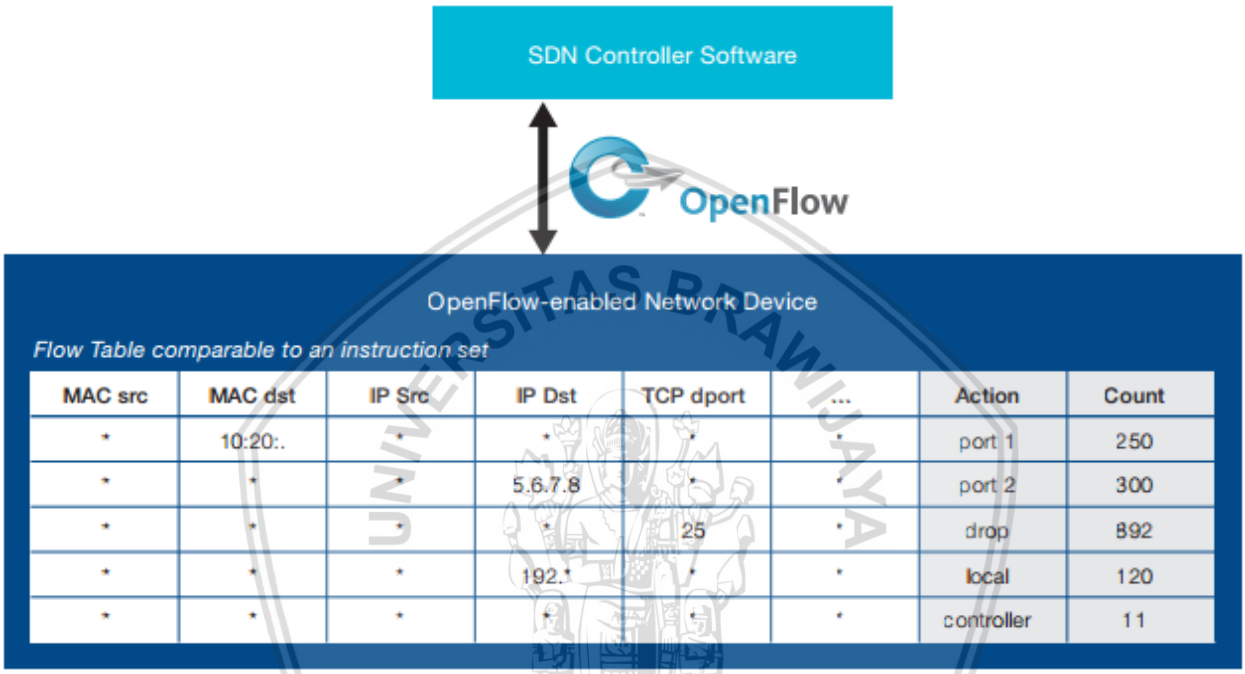
Gambar 2.3. Pemrosesan flow pada open-vSwitch.

Sumber : Open vSwitch Documentation (Open vSwitch, 2016) - http://openvswitch.org/

Gambar 2.3 menjelaskan alur kerja pengolahan *flow openvswitch*. Ketika sebuah *flow* tiba, modul *kernel openvswitch* akan menerima *flow* dari *ethernet-card* dan mencoba untuk menemukan *rule-flow* sesuai dalam tabel *flow* pada *kernel*. Jika ada *flow* yang sesuai dengan *rule-flow*, modul *kernel* akan mentransfer *flow* ini ke *user-space daemon* melalui soket *netlink*. Di sisi *user-space*, *daemon* secara berkala memeriksa paket yang masuk pada *netlink buffer*. Kemudian, *daemon* akan memeriksa tabel *flow* pada *user-space* untuk menemukan *rule-flow* yang cocok. Perbedaan utama antara tabel *flow* pada *user-space* dan tabel *flow* pada *kernel* adalah pemeriksaan kesesuaian. *flow* dalam modul *kernel* harus persis sesuai *rule-flow* dalam tabel *flow kernel* sementara *flow* di *daemon* hanya perlu memeriksa *rule-flow wildcard* pada tabel *flow* user-space. Jika *flow* masih tidak dapat sesuai dengan *rule-flow* dalam tabel *flow* pada *user-space*, *daemon* akan menghubungi *controller* untuk menanyakan tindakan yang akan dilakukan terhadap *flow* tersebut.

2.5. Open*flow*

Openflow adalah standar metode komunikasi antara *data-plane* dan *control-plane* yang pertama kali pada arsitektur jaringan SDN. *Openflow* memungkinkan untuk mengakses secara langsung dan melakukan perubahan konfigurasi pada perangkat *data-plane*. *Openflow* memerlukan sebuah *controller* terpusat untuk mengatur perangkat *data-plane* yang ada pada jaringan.



Gambar 2.4. Contoh instruksi dalam openflow.

Sumber : OpenFlow Documentation (OpenFlow 2011) - http://archive.openflow.org/

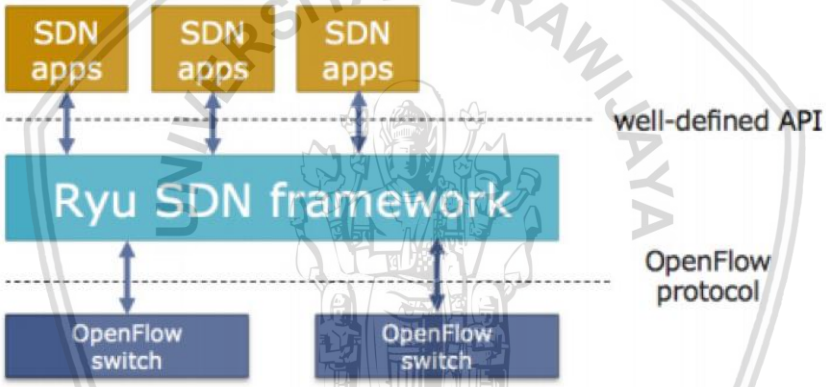
Openflow dapat dibandingkan dengan set instruksi CPU. Seperti ditunjukkan dalam gambar 2.4, protokol menentukan *rule* dasar yang dapat digunakan oleh sistem pada *forwarding device*, seperti set instruksi dari CPU akan memprogram sistem komputer (Open Networking Foundation, 2012).

Protokol *openflow* diimplementasikan di kedua sisi antara *data-plane* dan *controller* SDN. *Openflow* menggunakan konsep *flow* untuk mengidentifikasi trafik jaringan berdasarkan aturan/*rule* yang telah ditentukan secara statis atau dinamis yang diprogram oleh perangkat lunak *controller* SDN. Hal ini juga memungkinkan untuk menentukan bagaimana lalu lintas harus mengalir melalui perangkat jaringan berdasarkan parameter seperti pola penggunaan, aplikasi, dan *resource*. Sebuah arsitektur SDN berbasis *Openflow* memberikan kontrol yang sangat granular, memungkinkan jaringan untuk menanggapi perubahan *real-*

time di tingkat aplikasi, pengguna, dan sesi. Saat metode *routing* berbasis IP tidak memberikan tingkat kontrol, karena semua aliran antara dua *end-point* harus mengikuti jalan yang sama melalui jaringan, terlepas dari kebutuhan mereka berbeda.

2.6. Ryu

Ryu Controller adalah *open source* SDN *controller*. *Ryu* menyediakan komponen *software* yang didefinisikan dengan baik, program aplikasi antarmuka (API) yang membuatnya mudah bagi pengembang untuk membuat manajemen jaringan dan mengkontrol aplikasi. Pendekatan komponen ini membantu menyesuaikan penyebaran *flow* untuk memenuhi kebutuhan spesifik. Pengembang dapat dengan cepat dan mudah memodifikasi komponen yang ada atau menerapkan sendiri untuk memastikan jaringan dapat memenuhi tuntutan perubahan aplikasi mereka.



Gambar 2.5. Ryu pada arsitektur jaringan SDN.

Sumber : Ryu Documentation (Ryu, 2017) - https://osrg.github.io/ryu/

Pada Gambar 2.5 merupakan posisi *Ryu* dalam arsitektur jaringan SDN. *Ryu* berkomunikasi dengan SDN *application* dan *switch*. *Switch* pada gambar di atas menggunakan protokol *openflow*. *Ryu* dibangun dengan menggunakan bahasa pemrograman *Python*, semua kode *Ryu* tersedia di bawah lisensi *Apache* 2.0 dan terbuka bagi siapa saja untuk digunakan. *Ryu* mendukung *netconf* dan protokol manajemen jaringan *OF-config*, serta *openflow* yang merupakan salah satu standar komunikasi SDN pertama dan paling banyak digunakan.

Ryu dapat menggunakan *openflow* untuk berinteraksi dengan *forwarding plane* (*switch* dan *router*) untuk memodifikasi bagaimana jaringan akan menangani trafik *flow*. Ini telah diuji dan disertifikasi untuk dapat bekerja dengan sejumlah *openflow switch*, termasuk *open-vSwitch, Centec, Hewlett Packard, IBM* dan *NEC* (SDX Central, 2015).

Berikut *script* untuk instalasi *Ryu* pada Ubuntu (olegslavkin, 2016):

```
#!/bin/bash
# Install RYU (Ubuntu 14.04.3)
# Author: Oleg Slavkin

echo "Step 1. Install tools"
sudo apt-get -y install git python-pip python-dev

echo "Step 2. Install python packages"
sudo apt-get -y install python-eventlet python-routes python-webob
python-paramiko

echo "Step 3. Clone RYU git Repo"
cd ~/
git clone --depth=1 https://github.com/osrg/Ryu.git

echo "Step 4. Install RYU"
sudo pip install setuptools --upgrade
cd Ryu;
sudo python ./setup.py install

echo "Step 5. Install and Update python packages"
sudo pip install six --upgrade
sudo pip install oslo.config msgpack-python
sudo pip install eventlet --upgrade

echo "Step 6. Test Ryu-manager"
Ryu-manager -version
```

Berikut sintaks untuk meng-*import* modul *Ryu* pada pemrograman *python* (RYU, 2014) :

```
from Ryu.base import app_manager
from Ryu.controller import ofp_event
from Ryu.controller.handler import CONFIG_DISPATCHER,
MAIN_DISPATCHER, DEAD_DISPATCHER
from Ryu.ofproto.ofproto_v1_3 import OFPG_ANY
from Ryu.ofproto.ofproto_v1_3 import OFP_VERSION
from Ryu.controller.handler import set_ev_cls
from Ryu.ofproto import ofproto_v1_3
from Ryu.lib.packet import packet
from Ryu.lib.packet import ethernet
from Ryu.lib.packet import ether_types
from Ryu.lib.packet import arp
from Ryu.lib.packet import ipv4
from Ryu.lib.packet import tcp
from Ryu.lib.packet import udp
from Ryu.app.wsgi import ControllerBase, WSGIApplication, route
from Ryu.lib import dpid as dpid_lib
from Ryu.topology import event, switches
from Ryu.topology.api import get_switch, get_link, get_host
```

Untuk mendeteksi setiap *switch* yang terkoneksi dengan *controller*, *Ryu* menyediakan *event handler switch*. Berikut penulisan sintaks dalam bahasa pemrograman *python* :

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
      datapath = ev.msg.datapath
```

Untuk menginstal *flow-table* ke *switch* menggunakan contoh sintaks seperti berikut :

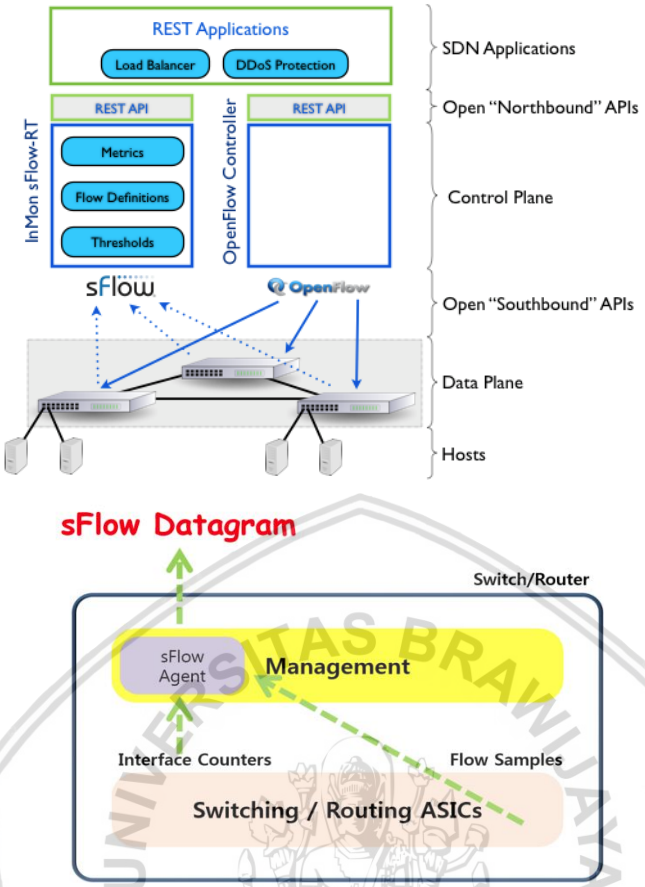
```
dpid = 1
dtFlowMatch = {}
dtFlowMatch['eth_type'] = 0x0800
dtFlowMatch['ipv4_src'] = '10.0.2.1'
dtFlowMatch['ipv4_dst'] = '10.0.2.10'
dtFlowMatch['ip_proto'] = 17
flowPriority = 103
pFlowMatch = parser.OFPMatch(**dtFlowMatch)
pFlowAction = []
outPort = 4
pFlowAction.append(parser.OFPActionOutput(outPort, 0))
self.addFlow(dpid, 0, flowPriority, pFlowMatch, pFlowAction)
```

2.7. WSGI

Web Server Gateway Interface (WSGI) adalah tata cara bagaimana sebuah *server web* berkomunikasi dengan aplikasi lainnya, dan bagaimana *web server* dapat bekerja sama untuk melayani permintaan (*request*) dalam rentetan proses sebuah sistem (WSGI.org, 2016). Bahasa pemrograman *Python* juga mendukung dalam penerapan WSGI. Metode akses WSGI sama seperti pada *web server* yaitu melalui HTTP *request*. *Request* yang dilakukan dapat menyertakan beberapa parameter. Penyertaan parameter dalam *request* dapat dilakukan dengan metode POST atau GET. Format data yang diberikan sebagai balasan *request* dapat berupa *plain text*, CSV atau JSON.

2.8. S*Flow*-RT

sFlow-RT adalah sebuah *tools* yang banyak digunakan untuk memproses *sFlow* paket yang diterima dari perangkat jaringan. Hal ini memungkinkan visibilitas *real-time* dari sebuah jaringan. *sFlow-RT* juga berjalan di *control-plane* dan *data-plane*. *sFlow-RT* merubah datagram menjadi metrik atau ringkasan statistik berdasarkan *flow* seperti yang didefinisikan oleh pengguna. Trafik *flow* diamati dalam jangka waktu tertentu. Identifikasi *flow* biasanya ditentukan oleh *field* dari *header* paket, seperti IP *source/destination* dan *port number* TCP / UDP. Data metrik statistik dapat diakses dengan menggunakan API. API yang mendukung pesan permintaan HTTP (*Perl*, *Python*, *Java*, *Java script*, *bash*, dll) dapat digunakan untuk mengambil metrik dari *sFlow-RT*. Statistik *sFlow-RT* dapat diambil dalam format JSON (Shafqat Ur Rehman, 2014). Contoh URL untuk mengambil data metrik dengan format JSON dari *sFlow-RT*: http://*127.0.0.1*:8008/*metric*/*agents*/*metrics*/json?filter.



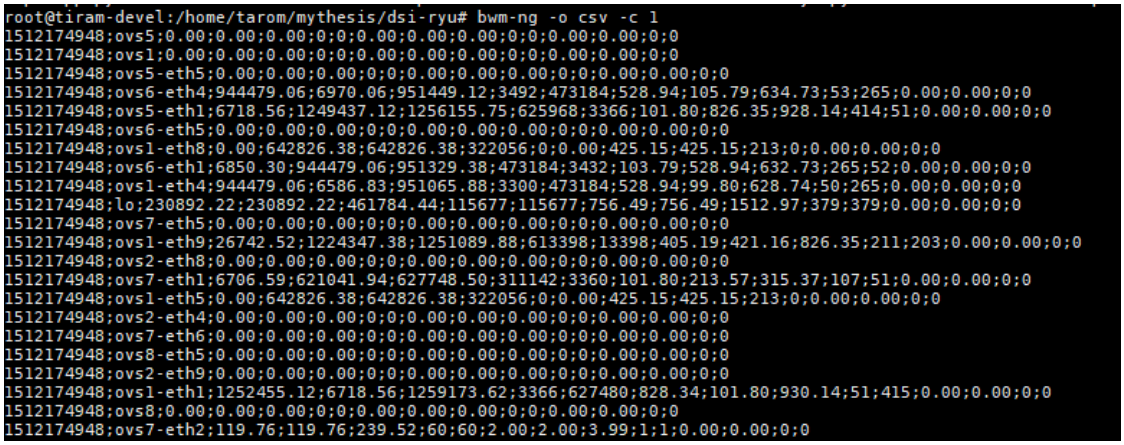
Gambar 2.6. sFlow-RT dalam arsitektur SDN.

Sumber : sFlow-RT (sFlow-RT, 2015) - http://sflow-rt.com/

Pada gambar 2.6 dijelaskan letak fungsional *sFlow-RT* pada arsitektur jaringan SDN. *SFlow-RT* dijalankan pada *controller* dan *switch*. Pada *switch* disebut *sFlow-agent* yang bertugas untuk melaporkan *sFlow* datagram pada *controller*. *sFlow datagram* kemudian diterima oleh *controller* melalui *sFlow-collector* yang berjalan pada *controller*.

2.9. Bwm-ng

Bandwidth monitor NG (*bwm-ng*) adalah sebuah aplikasi berbasis *command line* pada Linux untuk membaca *bandwidth* pada *network interface*. *Bwm-ng* mendukung format CSV untuk data hasil *monitoring*.



Gambar 2.7. Perintah dan output bwm-ng dengan format CSV

Pada gambar 2.7 ditunjukkan bagaimana untuk menggunakan *bwm-ng* untuk membaca data pada *network interface* dengan *output* dalam format CSV (Gropp, 2015). Data dalam format CSV terebut dapat dibaca dengan urutan data sebagai berikut :

1. *unix timestamp*
2. *interface*
3. *bytes_out/s*
4. *bytes_in/s*
5. *bytes_total/s*
6. *bytes_in*
7. *bytes_out*
8. *packets_out/s*
9. *packets_in/s*
10. *packets_total/s*
11. *packets_in*
12. *packets_out*
13. *errors_out/s*
14. *errors_in/s*
15. *errors_in*
16. *errors_out*

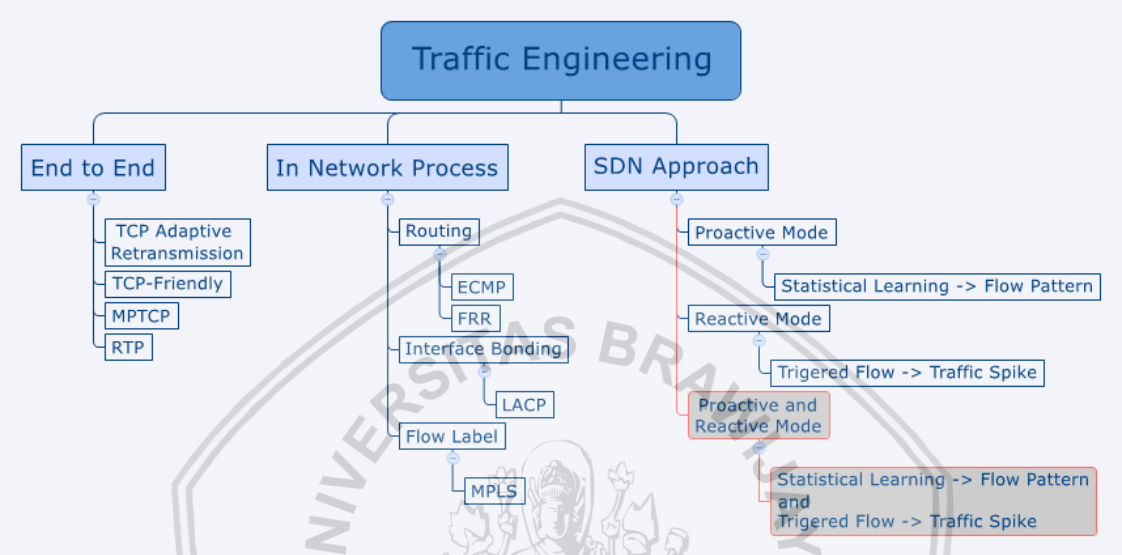
2.10. Mininet

Mininet adalah *emulator* untuk membuat dan menjalankan suatu model jaringan secara *virtual*. *Mininet* dilengkapi dengan fitur API (*Application Programming Interface*)

yang memudahkan pengguna untuk memanipulasi jaringan yang dibentuk. API juga memudahkan untuk para *developer* untuk mengintegrasikan kepada *external-application*. *Mininet* juga dapat menjalankan *switch open-vSwitch* dengan protokol *openflow*. Dilengkapi dengan modul *controller* yang dapat mengintegrasikan *external-controller* kepada *switch* pada model jaringan yang dijalankan. Dengan demikian jaringan yang dijalankan oleh *mininet* dapat dengan mudah diintegrasikan dengan *Ryu* sebagai SDN *controller*. Untuk memudahkan para peneliti, *mininet* juga menyediakan fitur limitasi dari I/O (*input*/*output*) *interface* jaringan yang dijalankan, sehingga peneliti tidak harus membuat *traffic* yang besar untuk mensimulasikan kepadatan jaringan (http://mininet.org/, 2016).

BAB III
KERANGKA KONSEP PENELITIAN

3.1. Kerangka Penelitian



Gambar 3.1. Kerangka penelitian

Disiplin ilmu pada penelitian ini adalah *Traffic Engineering*. Gambar 3.1 adalah gambar kerangka penelitian pada disiplin ilmu *Traffic Engineering*. Dimana pada disiplin ilmu ini banyak yang telah dilakukan oleh penelitian ataupun teknologi sebelumnya. *Traffic Engineering* adalah suatu metode yang bertujuan untuk mengoptimalkan kinerja jaringan dengan melakukan cara menganalisa dinamika jaringan, memprediksi dan mengatur perilaku data yang melewati jaringan tersebut (Fortz, 2002).

End to End adalah suatu metode *traffic engineering* yang dilakukan pada *source node* dan *destination node*. Kedua *node* melakukan negosiasi untuk mengatur trafik antara mereka. Beberapa contohnya adalah TCP *Adaptive Retransmission*, TCP-*Friendly*, *MP*TCP *(Multi Path* TCP*) dan* RTP *(Real-time Transport Protocol).* Pada TCP *Adaptive Retransmission*, setiap kali segmen TCP ditransmisikan, salinan itu juga ditempatkan pada antrian transmisi. Ketika segmen ditempatkan pada antrian, *timer retransmission* dimulai untuk segmen, yang dimulai dari nilai tertentu dan menghitung mundur ke nol. *Timer* ini yang mengontrol berapa lama segmen dapat tetap diakui sebelum dianggap *timeout*, menyimpulkan bahwa itu hilang dan mengirimkan ulang (The TCP/IP Guide, 2005). TCP-

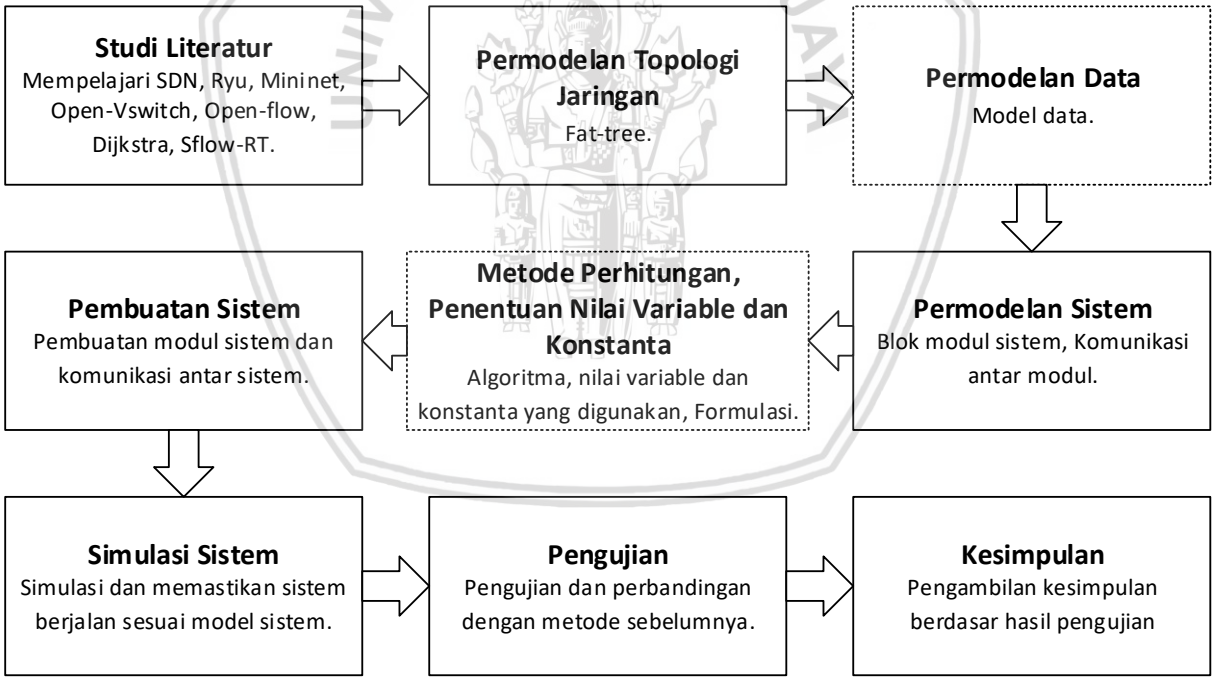
Friendly mengatur kongesti trafik selain TCP dalam sebuah jaringan sehingga diperlakukan sama seperti trafik TCP (The ICSI Networking and Security Group, 1999). Pada MPTCP memungkinkan untuk menggunakan beberapa IP *address* dengan memodifikasi paket TCP sehingga terlihat seperti paket TCP pada umumnya (MultiPath TCP - Linux Kernel implementation, 2012). Sedangkan pada RTP, komunikasi *end-to-end* untuk aplikasi transmisi data *real-time* seperti *audio*, *video*, *multicast* atau *unicast* (Internet Engineering Task Force, 2003).

Traffic engineering juga dilakukan pada sisi *network*. Pada sisi *network* terdapat *routing, interface bonding* dan *flow label* yang memungkinkan untuk dilakukan *traffic engineering*. Pada *routing* diantaranya ECMP *(Equal Cost Multi Path)* dan FRR *(Fast Reroute)*. ECMP adalah suatu metode *routing* pada jaringan *multi path* yang berfungsi untuk memutuskan ke *path* mana paket harus diteruskan berdasar *path-cost* pada tiap – tiap *path* pada jaringan tersebut (Internet Engineering Task Force, 2000). FRR adalah teknologi yang digunakan pada jaringan MPLS (*Multi Protocol Label Switching*) untuk memberikan jalur *routing* yang berbeda secara cepat ketika terjadi kegagalan pada *routing* yang digunakan (Internet Engineering Task Force, 2005).

SDN hadir dengan menyediakan fasilitas yang sangat fleksibel untuk melakukan *traffic engineering*. Pada SDN ada dua cara untuk melakukan *traffic engineering* yaitu *proactive mode* dan *reactive mode*. *Proactive mode* yang dimaksud adalah dimana *controller* akan menganalisa statistik jaringan selama beberapa waktu dan melakukan modifikasi perilaku jaringan berdasar data statistik tersebut. Jika perilaku jaringan berubah drastis dari rentang waktu statistik sebelumnya, maka *controller* tidak melakukan perubahan secara langsung, melainkan menunggu rentang statistik selesai dan menganalisa berdasar statistik terbaru. Sedangkan pada *reactive mode*, *controller* akan melakukan *monitoring* jaringan secara *real-time* dan mengidentifikasi setiap *traffic-flow* yang diterima oleh *switch*, kemudian *controller* akan memberlakukan *traffic-flow* tersebut sesuai dengan data *monitoring*. *Proactive mode* dalam hal kecepatan *forwarding flow* lebih bagus dari pada *reactive mode*. Karena pada *proactive mode* hasil analisa statistik akan diterapkan kepada seluruh *switch* sehingga *switch* tidak perlu menghubungi *controller* untuk memutuskan ke mana *flow* harus diteruskan. Akan tetapi sensitifitas terhadap *traffic-spike* atau perubahan perilaku trafik pada jaringan dinilai lambat, karena memang pada mode ini harus menunggu data statistik terkumpul dengan rentang waktu yang telah ditentukan. *Reactive mode* sangat

sensitif terhadap perubahan perilaku trafik pada jaringan. Akan tetapi karena terlalu sensitif memungkinkan terjadinya *isolation-process* pada penerapannya. *Isolation-process* yang dimaksud adalah menemui keadaan dimana harus dilakukan perhitungan ulang dan menerapkan secara berkali – kali dalam jarak waktu yang dekat. *Reactive mode* untuk kecepatan *forwarding flow* juga lebih lambat dari pada *proactive mode*. Hal ini terjadi karena pada *reactive mode* tiap *flow* baru (yang belum teridentifikasi sebelumnya), *switch* akan berkomunikasi kepada *controller* untuk memutuskan ke mana *flow* tersebut akan diteruskan. Pada penelitian ini kedua metode tersebut akan dikombinasikan untuk melakukan *traffic engineering*, dimana *controller* akan melakukan statistik pada rentang waktu secara terus menerus sekaligus melakukan *monitoring* secara *real-time* perubahan yang terjadi pada jaringan, sehingga *controller* dapat melakukan perubahan perilaku jaringan baik secara *proactive* maupun secara *reactive*.

3.2. Alur Langkah Penelitian



Gambar 3.2. Alur langkah penelitian.

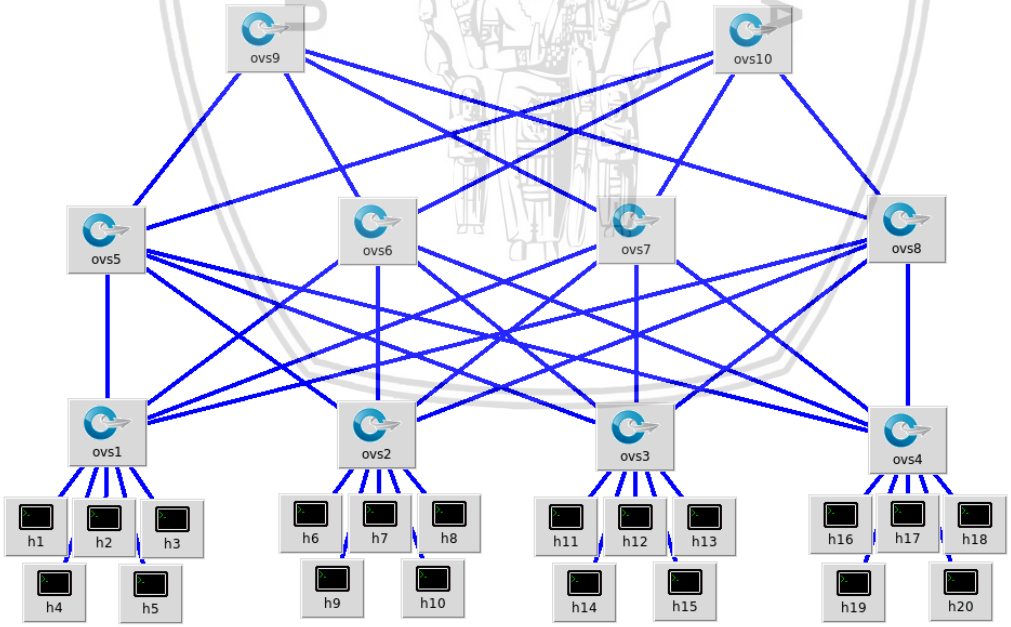
Untuk memastikan penelitian berjalan sesuai dengan kerangka konsep penelitian, maka pada Gambar 3.2 dapat dilihat alur langkah penelitian yang akan dilakukan dalam penelitian ini.

3.2.1. Studi Literatur

Tahap pertama adalah melakukan studi literatur, yaitu dengan mengumpulkan dan mempelajari literatur, jurnal, buku, artikel, dan sebagainya yang diperoleh dari perpustakaan, *internet*, dan sumber lainnya mengenai SDN, *dynamic-routing*, *open-vSwitch*, *open-Flow*, *Dijkstra* dan materi-materi lain yang dibutuhkan dalam penyusunan laporan tesis. Jadi, studi literatur diperlukan untuk mendapatkan dasar teori dan merancang kerangka konsep penelitian yang berkaitan dengan penelitian yang akan dilakukan

3.2.2. Permodelan Topologi Jaringan

Berikut adalah topologi jaringan yang akan digunakan dalam penelitian ini. Topologi yang digunakan adalah topologi *fat-tree* dengan 4 *switch ToR* (*Top of Rack*) yaitu ovs1, ovs2, ovs3 dan ovs4. Masing – masing *ToR switch* terhubung secara *mesh* pada 4 *switch* agregasi (ovs5, ovs6, ovs7 dan ovs8). Kemudian tiap – tiap *switch* agregasi terhubung secara *mesh* juga pada 2 *core switch* (ovs9 dan ovs10). Pada tiap – tiap *rack* terdapat 5 *host* yang terkoneksi secara langsung kepada *switch ToR rack* tersebut.



Gambar 3.3. Topologi jaringan.

Sumber : topologi fat-tree (Hui Long, 2013)

Gambar 3.3 di atas adalah tampilan visual dari topologi yang digunakan pada penelitian ini. *Source code* untuk membangun topologi seperti pada Gambar 3.3 dalam

mininet terlampir. Sedangkan pemberian identitas unik (*dpid*) pada setiap *switch* seperti pada tabel 3.1.

Tabel 3.1. Tabel switch dpid

Switch	***dpid***
ovs1	1
ovs2	2
ovs3	3
ovs4	4
ovs5	5
ovs6	6
ovs7	7
ovs8	8
ovs9	9
ovs10	10

Pada tabel 3.1 *switch* ovs1 hingga ovs10 diberikan *dpid* secara berurutan mulai 1 hingga 10. Hal ini dilakukan untuk memudahkan dalam proses penelitian.

Tabel 3.2. Pengalamatan host

Host	***IP Address***
h1	10.0.2.1
h2	10.0.2.2
h3	10.0.2.3
h4	10.0.2.4
h5	10.0.2.5
h6	10.0.2.6
h7	10.0.2.7
h8	10.0.2.8
h9	10.0.2.9
h10	10.0.2.10
h11	10.0.2.11
h12	10.0.2.12
h13	10.0.2.13
h14	10.0.2.14
h15	10.0.2.15
h16	10.0.2.16
h17	10.0.2.17
h18	10.0.2.18
h19	10.0.2.19
h20	10.0.2.20

Tabel 3.2 merupakan daftar pengalamatan atau pemberian IP *address* dari setiap *host* yang terkoneksi dalam topologi penelitian.

3.2.3. Permodelan Data

Data dikelompokkan menjadi dua kelompok, yaitu :

1. Data yang dibutuhkan sebelum sistem dijalankan.
   Meliputi data *host*, data *switch*, data *network interface* dan data *path* jaringan.
2. Data yang ada setelah sistem dijalankan.
   Meliputi data *path-load*, data *overload-flow*, data segmentasi *traffic-flow* dan data *flow-table*.

3.2.3.1. Data *host* (tabel `*t_host*`)

Tabel 3.3. Data host.

id	*hw_addr*	ipv4	label
ba:6b:20:c8:0e:ec	ba:6b:20:c8:0e:ec	10.0.2.2	2.2
9a:61:df:09:41:8f	9a:61:df:09:41:8f	10.0.2.3	2.3
4a:fd:4a:c8:dd:45	4a:fd:4a:c8:dd:45	10.0.2.1	2.1
22:d7:43:18:7f:49	22:d7:43:18:7f:49	10.0.2.4	2.4
6e:09:2c:0e:54:b8	6e:09:2c:0e:54:b8	10.0.2.5	2.5
ea:11:33:af:e9:c5	ea:11:33:af:e9:c5	10.0.2.9	2.9
7e:cb:f3:1b:1c:c3	7e:cb:f3:1b:1c:c3	10.0.2.10	2.1
46:2e:df:c2:01:77	46:2e:df:c2:01:77	10.0.2.6	2.6
92:50:29:24:f6:2d	92:50:29:24:f6:2d	10.0.2.8	2.8
9e:00:14:7d:a1:18	9e:00:14:7d:a1:18	10.0.2.7	2.7
b2:0c:c7:14:46:f5	b2:0c:c7:14:46:f5	10.0.2.15	2.15
26:3f:44:c0:2f:6d	26:3f:44:c0:2f:6d	10.0.2.14	2.14
8a:4a:74:e1:09:c6	8a:4a:74:e1:09:c6	10.0.2.11	2.11
be:e3:92:aa:a2:cc	be:e3:92:aa:a2:cc	10.0.2.12	2.12
ca:5e:93:5e:b5:69	ca:5e:93:5e:b5:69	10.0.2.13	2.13
66:92:54:a7:d6:04	66:92:54:a7:d6:04	10.0.2.20	2.2
3a:47:67:5e:af:dc	3a:47:67:5e:af:dc	10.0.2.18	2.18
c2:11:13:c4:d8:67	c2:11:13:c4:d8:67	10.0.2.16	2.16
c6:4e:63:d2:00:42	c6:4e:63:d2:00:42	10.0.2.17	2.17
6a:d5:e5:da:5b:be	6a:d5:e5:da:5b:be	10.0.2.19	2.19

Tabel 3.3 adalah struktur kolom dan data tabel *t_host* yang memuat data *host* yang terkoneksi kepada masing – masing *switch*. Data *host* meliputi MAC *address network*

interface (kolom *hw_addr*), IP *address* (kolom ipv4) dan label penanda agar lebih mudah untuk proses pembacaan (kolom label). Kolom yang dijadikan *primary key* adalah kolom *id* yang datanya sama dengan kolom *hw_addr*. Tabel 3.4 adalah struktur data dari tabel *t_host*.

Tabel 3.4. Struktur data tabel t_host

Field	Type	Key
id	varchar(30)	PRI
hw_addr	varchar(30)	
ipv4	varchar(30)	
label	varchar(100)	

Pada Tabel 3.4 nampak bahwa hanya satu kolom yang dijadikan sebagai *primary key* yaitu kolom *id* yang berisikan data MAC *address* pada *network interface* yang digunakan *host* tersebut.

3.2.3.2. Data *switch* (tabel `*t_switch*`)

Tabel 3.5. Data switch.

id	*dpid*	*sflow_agent*	label
1	1	127.0.0.1	ovs1
2	2	127.0.0.1	ovs2
3	3	127.0.0.1	ovs3
4	4	127.0.0.1	ovs4
5	5	127.0.0.1	ovs5
6	6	127.0.0.1	ovs6
7	7	127.0.0.1	ovs7
8	8	127.0.0.1	ovs8
9	9	127.0.0.1	ovs9
10	10	127.0.0.1	ovs10

Tabel 3.5 adalah struktur kolom dan data tabel *t_switch* yang memuat data semua *switch* yang terkoneksi dengan *controller*. Data *switch* meliputi *datapath identity* (kolom *dpid*) yang merupakan identitas unik dari setiap *switch* dalam berkomunikasi dengan *controller*. Selain itu juga terdapat kolom *agent_id* yang memuat data identitas yang dikenali pada sistem *monitoring flow* menggunakan *sFlow-RT*. Tampak pada kolom *agent_id* terisikan 127.0.0.1 ini dikarenakan sistem *monitoring sFlow-RT* dan *mininet* dijalankan pada *machine* (komputer/*server*) yang sama. Sedangkan kolom *label* merupakan suatu penandaan dengan tujuan memudahkan pembacaan pada sistem. Kolom yang dijadikan *primary key*

adalah kolom *id* yang datanya sama dengan kolom *dpid*. Tabel 3.6 adalah struktur data dari tabel *t_switch*.

Tabel 3.6. Struktur data tabel t_switch

Field	Type	Key
id	varchar(30)	PRI
dpid	bigint(20)	
s*flow*_agent	varchar(100)	
label	varchar(100)	

Pada Tabel 3.6 nampak bahwa hanya satu kolom yang dijadikan sebagai *primary key* yaitu kolom *id* yang berisikan data *dpid* dari *switch* tersebut.

3.2.3.3. Data *interface* (tabel `*t_intf*`)

Tabel 3.7. Data interface.

id	*hw_addr*	*name*	*dpid*	*port_no*	*os_index*	*os_name*
86:0f:61:a3:2b:79	86:0f:61:a3:2b:79	ovs1-eth1	1	1	1958	ovs1-eth1
9e:d5:20:2f:38:d6	9e:d5:20:2f:38:d6	ovs1-eth2	1	2	1960	ovs1-eth2
32:11:72:0e:85:50	32:11:72:0e:85:50	ovs1-eth3	1	3	1962	ovs1-eth3
c6:c9:d9:6e:97:e2	c6:c9:d9:6e:97:e2	ovs1-eth4	1	4	1964	ovs1-eth4
32:4c:cc:70:37:49	32:4c:cc:70:37:49	ovs1-eth5	1	5	2005	ovs1-eth5
a6:6d:21:a9:03:42	a6:6d:21:a9:03:42	ovs1-eth6	1	6	2006	ovs1-eth6
ae:c9:9c:18:bf:e2	ae:c9:9c:18:bf:e2	ovs1-eth7	1	7	2007	ovs1-eth7
66:ec:dc:e3:65:40	66:ec:dc:e3:65:40	ovs1-eth8	1	8	2008	ovs1-eth8
8a:9f:1f:79:88:58	8a:9f:1f:79:88:58	ovs1-eth9	1	9	2009	ovs1-eth9
ba:6b:20:c8:0e:ec	ba:6b:20:c8:0e:ec	\N	\N	\N	\N	\N
9a:61:df:09:41:8f	9a:61:df:09:41:8f	\N	\N	\N	\N	\N
4a:fd:4a:c8:dd:45	4a:fd:4a:c8:dd:45	\N	\N	\N	\N	\N
22:d7:43:18:7f:49	22:d7:43:18:7f:49	\N	\N	\N	\N	\N
6e:09:2c:0e:54:b8	6e:09:2c:0e:54:b8	\N	\N	\N	\N	\N

Tabel 3.7 adalah struktur kolom dan data tabel *t_intf* yang memuat data *interface* pada masing – masing *switch* dan *host*. Data *interface* meliputi MAC *address network interface* tersebut yang dimuat dalam tabel *hw_addr*. Kolom *name* berisikan data nama *interface* yang terbaca pada sistem *monitoring sFlow-RT*. Nama *interface* untuk *interface* pada *host* berisikan N/A (*not available*), ini dikarenakan pada penelitian ini *sFlow-RT agent* tidak dijalankan pada tiap – tiap *host*. Akan tetapi data *interface* pada *host* sudah cukup diwakili oleh MAC *address* yang nantinya dapat dipetakan kepada IP *address* yang

digunakan oleh *host* tersebut. Kolom *dpid* berisi *dpid* dari *switch* jika *interface* tersebut adalah *interface* dari *switch*. Kolom *port_no* berisi penomoran *interface* pada *open-vSwitch* yang digunakan dalam *protocol openflow*. Kolom *os_index* berisi data *index* atau penomoran dari *virtual interface mininet* yang terbaca pada OS (*Operating System*). Data *os_index* ini adalah sebagai data kunci untuk pemetaan antara data *monitoring sFlow-RT* dan data pada *mininet*. Kolom *os_name* berisi data nama *virtual network interface* dalam *mininet* yang terbaca pada OS (*Operating System*). Data *os_name* digunakan untuk memetakan hasil *monitoring* menggunakan *bwm-ng*, dikarenakan *bwm-ng* menggunakan nama *interface* yang terdaftar pada OS untuk memberikan nilai beban *traffic-flow* pada *interface* tersebut. Tabel 3.8 adalah struktur data dari tabel *t_intf*.

Tabel 3.8. Struktur data tabel t_intf

Field	Type	Key
id	varchar(30)	PRI
hw_addr	varchar(30)	
name	varchar(30)	
dpid	bigint(30)	
port_no	bigint(20)	
os_index	int(11)	
os_name	varchar(100)	
bytes_in_s	bigint(20)	
bytes_out_s	bigint(20)	
*bytes_in*_udp	bigint(20)	
tcp_*flow*_count	bigint(20)	

Pada Tabel 3.8 nampak bahwa hanya satu kolom yang dijadikan sebagai *primary key* yaitu kolom *id* yang berisikan data MAC *address* dari *network interface* tersebut.

3.2.3.4. Data *link* jaringan (tabel `*t_link*`)

Tabel 3.9. Data path.

src_intf_id	***dst_intf_id***	**bw**
a6:6d:21:a9:03:42	ba:6b:20:c8:0e:ec	0
ba:6b:20:c8:0e:ec	a6:6d:21:a9:03:42	0
72:34:cc:52:c8:07	7a:8e:89:39:36:ff	10000000
52:5c:1c:65:0a:e6	6a:f2:5d:92:57:c3	20000000

Tabel 3.9 adalah struktur kolom dan data tabel *t_link* yang berisikan informasi tentang data *link* atau *path* koneksi antar *node* dalam jaringan. Data *path* terdiri dari dua

jenis. Yang pertama adalah *path* antara *switch* ke *host* dan sebaliknya, dan yang kedua adalah *path* antar *switch*. Satu koneksi antar *node* mempunyai dua *path*. Hal ini dikarenakan koneksi kabel mempunyai jalur *Tx* (pengiriman) dan *Rx* (penerimaan) yang terpisah. Dalam data *path* terdapat kolom *src_intf_id* dan *dst_intf_id* yang berisikan MAC *address* dari *interface* sumber dan *interface* tujuan. Untuk membedakan apakah *path* tersebut merupakan *path* anatar *switch* atau dari *switch* ke *host*, maka dapat dipetakan dengan referensi MAC *address* pada tabel *t_intf*. Kolom *bw* berisikan data *bandwidth* maksimum dari *path* tersebut. Limitasi *bandwidth* menggunakan fitur yang telah disediakan oleh *mininet*. Tabel 3.10 adalah struktur data dari tabel *t_link*.

Tabel 3.10. Struktur data tabel t_intf

Field	Type	Key
src_intf_id	varchar(50)	PRI
dst_intf_id	varchar(50)	PRI
bw	bigint(20)	

Pada Tabel 3.10 nampak bahwa hanya satu kolom yang dijadikan sebagai *primary key* yaitu kolom *src_intf_id* yang berisikan data MAC *address* dari *network interface* asal dan kolom *dst_intf_id* yang berisikan data MAC *address* dari *network interface* tujuan.

3.2.3.5. Data *interface load* (digabungkan dengan tabel `*t_intf*`)

Tabel 3.11. Data interface load.

id	*bytes_in_s*	*bytes_out_s*
86:0f:61:a3:2b:79	0	0
9e:d5:20:2f:38:d6	0	0
32:11:72:0e:85:50	0	0
c6:c9:d9:6e:97:e2	0	0
…	…	…
6a:f2:5d:92:57:c3	0	0

Tabel 3.11 adalah data beban *traffic-load* pada setiap *interface* yang digabungkan dengan tabel *t_intf*. Kolom yang ditambahkan adalah *bytes_in_s* dan *bytes_out_s*. Kolom tersebut berisikan kapasitas beban *traffic-flow* dalam satuan *bytes* per detik (bps). *Traffic-flow* yang dikirim disimpan pada kolom *bytes_out_s* dan *traffic-flow* yang diterima pada kolom *bytes_in_s*. Data tersebut diperoleh dari *output monitoring* menggunakan *bwm-ng*.

Karena data *interface load* ini digabungkan dengan data *interface*, maka struktur data tabel mengikuti struktur data tabel *t_intf*.

3.2.3.6. Data *overload flow* (tabel `t_*overload_flow*`)

Tabel 3.12. Data flow load.

time	*segment_id*	*src_ipv4*	*dst_ipv4*	proto	*src_proto_port*	*dst_proto_port*	*load*
2017-11-21 21:35:41	10.0.2.3/-1/17/3003/10.0.2.18	10.0.2.3	10.0.2.18	17	-1	3003	575634
2017-11-21 21:35:41	10.0.2.4/-1/17/3004/10.0.2.19	10.0.2.4	10.0.2.19	17	-1	3004	515695
2017-11-21 21:35:41	10.0.2.1/-1/17/3001/10.0.2.16	10.0.2.1	10.0.2.16	17	-1	3001	357947
2017-11-21 21:35:41	10.0.2.2/-1/17/3002/10.0.2.17	10.0.2.2	10.0.2.17	17	-1	3002	255177
2017-11-21 21:35:41	10.0.2.5/-1/6/2001/10.0.2.20	10.0.2.5	10.0.2.20	6	-1	2001	103494
2017-11-21 21:35:41	10.0.2.5/-1/6/2003/10.0.2.20	10.0.2.5	10.0.2.20	6	-1	2003	53578
2017-11-21 21:35:41	10.0.2.20/2001/6/-1/10.0.2.5	10.0.2.20	10.0.2.5	6	2001	-1	3279
2017-11-21 21:35:41	10.0.2.20/2003/6/-1/10.0.2.5	10.0.2.20	10.0.2.5	6	2003	-1	499

Tabel 3.12 adalah struktur kolom dan data tabel *t_overload_flow* yang menyimpan data *traffic-flow* pada seluruh *path* ketika terjadi *overload-path* pada salah satu atau beberapa *path* pada jaringan. Kolom *segment_id* adalah identitas segmen dari *traffic-flow*. Kolom *src_ipv4* adalah IP *address* pengirim dari *traffic-flow*. Kolom *dst_ipv4* adalah IP *address* tujuan dari *traffic-flow*. Kolom *protocol* berisikan *protocol* yang digunakan oleh *traffic-flow*. Kolom *src_proto_port* adalah *port number* yang digunakan pada sisi *node* pengirim dan *dst_proto_port* adalah *port number* yang digunakan pada sisi *node* penerima. *Port number* '-1' adalah penanda untuk *port number dynamic* pada OS. Kolom *load* berisi data nilai kapasitas dari segmen *traffic-load* tersebut. Tabel 3.13 adalah struktur data dari tabel *t_overload_flow*.

Tabel 3.13. Struktur data tabel t_overload_flow

Field	Type	Key
time	date*time*	PRI
segment*_id*_1	varchar(100)	
segment*_id*_2	varchar(100)	
src_ipv4	varchar(20)	PRI
dst_ipv4	varchar(20)	PRI
proto	int(11)	PRI
src_proto_port	int(11)	PRI
dst_proto_port	int(11)	PRI
load	bigint(11)	

Pada Tabel 3.13 nampak bahwa tabel *t_overload_flow* menggunakan 6 kolom sebagai *primary key* agar tidak terjadi duplikasi data dan kemudahan pengolahan data pada proses segmentasi.

3.2.3.7. Data segmentasi trafik (tabel `*t_traffic_segment*`)

Tabel 3.14. Data segmentasi *trafik*

id	*src_ipv4*	*src_proto_port*	proto	*dst_proto_port*	*dst_ipv4*	*load*
10.0.2.4/-1/17/3004/10.0.2.19	10.0.2.4	-1	17	3004	10.0.2.19	227503
10.0.2.2/-1/17/3002/10.0.2.17	10.0.2.2	-1	17	3002	10.0.2.17	237858
10.0.2.3/-1/17/3003/10.0.2.18	10.0.2.3	-1	17	3003	10.0.2.18	302094
10.0.2.1/-1/17/3001/10.0.2.16	10.0.2.1	-1	17	3001	10.0.2.16	328729
10.0.2.5/-1/6/2001/10.0.2.20	10.0.2.5	-1	6	2001	10.0.2.20	168207
10.0.2.5/-1/6/2002/10.0.2.20	10.0.2.5	-1	6	2002	10.0.2.20	32228
10.0.2.5/-1/6/2003/10.0.2.20	10.0.2.5	-1	6	2003	10.0.2.20	98374
10.0.2.5/-1/6/2004/10.0.2.20	10.0.2.5	-1	6	2004	10.0.2.20	227707
10.0.2.20/2004/6/-1/10.0.2.5	10.0.2.20	2004	6	-1	10.0.2.5	9705
10.0.2.20/2003/6/-1/10.0.2.5	10.0.2.20	2003	6	-1	10.0.2.5	4038
10.0.2.3/-1/-1/-1/10.0.2.5	10.0.2.3	-1	-1	-1	10.0.2.5	0
10.0.2.3/-1/-1/-1/10.0.2.4	10.0.2.3	-1	-1	-1	10.0.2.4	0
10.0.2.3/-1/-1/-1/10.0.2.20	10.0.2.3	-1	-1	-1	10.0.2.20	0
10.0.2.3/-1/-1/-1/10.0.2.6	10.0.2.3	-1	-1	-1	10.0.2.6	0

Tabel 3.14 adalah struktur kolom dan data tabel *t_traffic_segment* yang merupakan hasil dari pengelompokan atau segmentasi *traffic-flow* menjadi bagian – bagian yang kecil. Kolom *segment_id* merupakan identitas dari sebuah segmen *traffic-flow*. Identitas segmen *traffic-flow* merupakan gabungan dari IP *address* sumber dan tujuan, *internet protocol number*, dan *port number* yang digunakan. Kolom *src_ipv4*, *dst_ipv4*, *proto*, *src_proto_port* dan *dst_proto_port* sama halnya dengan data yang telah dijelaskan pada tabel *t_overload_flow*. Yang membedakan antara tabel *t_overload_flow* dengan tabel *t_traffic_segment* adalah kolom *time*. Pada tabel *t_overload_flow* adalah data *segment traffic-flow* yang terjadi pada saat terjadi *overload-path* dan data ini diambil tiap rentang waktu tertentu selama waktu *threshold overload-path*. Sedangkan pada tabel *t_traffic_segment* adalah data segmen *traffic-flow* hasil akumulasi dari seluruh data segmen *traffic-flow* yang terjadi selama *threshold* waktu *overload-path*. Tabel 3.15 adalah struktur data dari tabel *t_traffic_segment*.

Tabel 3.15. Struktur data tabel t_traffic_segment

Field	Type	Key
id	varchar(100)	PRI
src_ipv4	varchar(20)	
src_proto_port	int(11)	
proto	int(11)	
dst_proto_port	int(11)	
dst_ipv4	varchar(20)	
load	bigint(20)	

Pada Tabel 3.15 nampak bahwa hanya satu kolom yang dijadikan sebagai *primary key* yaitu kolom *id* yang berisikan data identitas pemetaan segmen (telah dibahas pada sub bab sebelumnya) dari data *traffic-flow* tersebut.

3.2.3.8. Data *flow*-table (tabel `t_*flow*_table`)

Tabel 3.16. Data flow.

id	segment_id	hop_index	switch_id	src_ipv4	dst_ipv4	proto	src_proto_port	dst_proto_port	out_port	priority
1	10.0.2.1/-1/17/3001/10.0.2.16	1	1	10.0.2.1	10.0.2.16	17	-1	3001	3	0
2	10.0.2.1/-1/17/3001/10.0.2.16	2	7	10.0.2.1	10.0.2.16	17	-1	3001	4	0
3	10.0.2.1/-1/17/3001/10.0.2.16	3	4	10.0.2.1	10.0.2.16	17	-1	3001	5	0
4	10.0.2.3/-1/17/3003/10.0.2.18	1	1	10.0.2.3	10.0.2.18	17	-1	3003	2	0
5	10.0.2.3/-1/17/3003/10.0.2.18	2	6	10.0.2.3	10.0.2.18	17	-1	3003	4	0

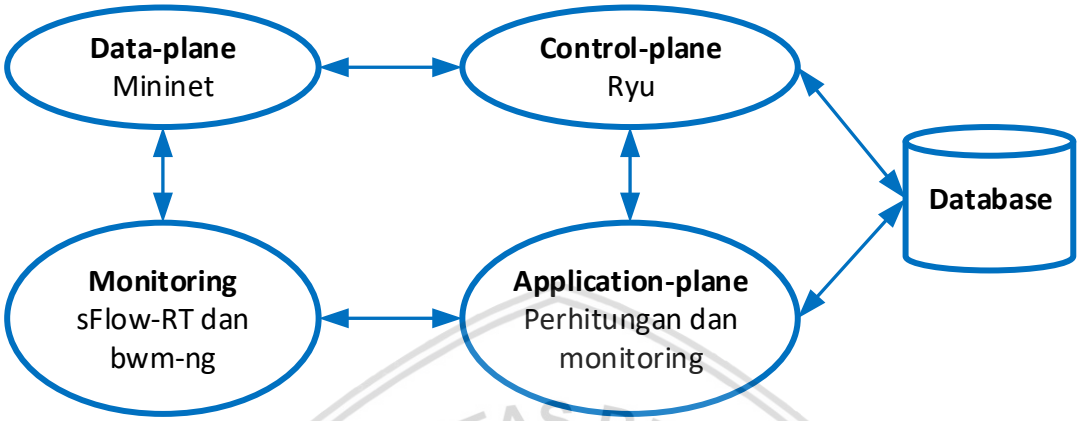
Tabel 3.16 adalah struktur kolom dan data tabel *t_flow_table*. Data *t_flow_table* inilah yang akan ditambahkan kepada masing – masing *switch*. Pada tabel ini setiap segmen *traffic-flow* mempunyai beberapa *record* data. Jumlah *record* tersebut melambangkan jumlah *switch* yang akan dilewati dari sebuah segmen *traffic-flow* dari pengirim menuju penerima. Kolom *hop_index* menandakan urutan *switch* yang akan dilalui dari suatu segmen *traffic-flow* pada kolom *segment_id*. Kolom *out_port* berisi data *port output* untuk meneruskan segmen *traffic-flow* pada *switch* kolom *switch_id*. . Berikut adalah struktur data dari tabel *t_flow_table* :

Tabel 3.17. Struktur data tabel t_flow_table

Field	Type	Key
id	int(11)	PRI
segment_id	varchar(100)	
hop_index	int(11)	
switch_id	varchar(11)	
src_ipv4	varchar(20)	
dst_ipv4	varchar(20)	
proto	int(11)	
src_proto_port	int(11)	
dst_proto_port	int(11)	
out_port	int(11)	
priority	int(11)	

Pada Tabel 3.17 nampak bahwa hanya satu kolom yang dijadikan sebagai *primary key* yaitu kolom *id* yang berisikan data *auto increment number*.

3.2.4. Permodelan Sistem



Gambar 3.4. Blok diagram modul sistem

Gambar 3.4 di atas menggambarkan hubungan antar modul yang digunakan pada penelitian ini. Pembahasan setiap modul akan dibahas pada sub bab berikutnya. Garis antar modul melambangkan komunikasi antar modul yang akan dibahas pada sub bab berikutnya.

3.2.4.1. Data-plane

Data-plane yang dimaksud adalah *switch* yang menggunakan *kernel open-vSwitch* dan menggunakan protokol *openflow*. Pada penelitian ini *switch*, *host* dan topologi jaringan di-emulasikan oleh *mininet*. Komunikasi dengan *controller* menggunakan protokol TCP *port number* 6633. Dalam berkomunikasi dengan *controller*, *switch* bertindak sebagai *socket-client*. Pada *switch* yang diemulasikan juga dijalankan s*Flow-agent*. s*Flow-agent* akan melakukan komunikasi kepada s*Flow-collector* (dalam hal ini sistem *monitoring sFlow-RT*) menggunakan protokol UDP *port number* 6343. Dimana dalam berkomunikasi dengan *sFlow-RT*, *data-plane* bertindak sebagai *socket-client*.

3.2.4.2. Control-plane

Control-plane pada penelitian ini menggunakan *Ryu* SDN *Controller*. Protokol yang digunakan untuk berkomunikasi dengan *switch* adalah *openflow*. Komunikasi dengan *switch* menggunakan protokol TCP *port number* 6633. Dalam berkomunikasi dengan *switch*, *controller* bertindak sebagai *socket-server*. Sedangkan komunikasi *control-plane* dengan *application plane* menggunakan protokol TCP *port number* 8080. Dalam berkomunikasi

dengan *application-plane*, *Ryu* bertindak sebagai *socket-server*. *Ryu* juga berkomunikasi dengan *database* untuk membaca data *flow-table* hasil proses perhitungan yang dilakukan *application-plane*. Komunikasi dengan *database* menggunakan *mysql-connector* yang berjalan dengan protokol TCP *port* 3306.

Dalam berkomunikasi dengan *application-plane Ryu* menggunakan modul WSGI (*Web Server Gateway Interface*) sebagai penyedia *web-service*. *Port number* 8080 adalah *port* standar dari *python* WSGI. WSGI inilah yang nantinya diakses oleh *application-plane* menggunakan HTTP *Request*. Konfigurasi WSGI pada *script Ryu* adalah sebagai berikut :

```
from Ryu.app.wsgi import ControllerBase, WSGIApplication, route

class MyDynRouteApp(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    _CONTEXTS = {'wsgi': WSGIApplication}

    def __init__(self, *args, **kwargs):
        super(MyDynRouteApp, self).__init__(*args, **kwargs)
        wsgi = kwargs['wsgi']

wsgi.register(MyDynRouteController,{MyDynRouteAppInstanceName:
self})

class MyDynRouteController(ControllerBase):

    def __init__(self, req, link, data, **config):
        super(MyDynRouteController, self).__init__(req, link, data,
**config)
        self.MyDynRouteApp = data[MyDynRouteAppInstanceName]
        self.mLogger = self.MyDynRouteApp.mLogger
```

Pada WSGI yang disediakan terdapat beberapa modul yang dapat diakses oleh *application-plane*. Berikut adalah daftar modul yang terdapat pada *control-plane* penelitian ini :

a. *discoveryNetwork* (http://127.0.0.1:8080/*controller*/*discoveryNetwork*)

   Modul ini diakses oleh *application plane* pada tahap inisialisasi (dibahas pada sub bab berikutnya). Modul ini akan memberikan nilai balik berupa data *switch*, *host*, *network interface* dan *link* pada jaringan. Data ini didapat dari *switch* saat terkoneksi dengan *control-plane*. Berikut adalah bagian *script* untuk modul *discoveryNetwork* (*script* secara lengkap terlampir) :

```
@route('controller', MyDynRouteAppBaseUrl+'discoveryNetwork')
def discoveryNetwork(self, req, **kwargs):
      self.mLogger.info('wsgi '+str(req))
      dtReply = {}
      dtReply = self.MyDynRouteApp.discoveryNetwork()
      body =json.dumps(dtReply)
      return Response(content_type='application/json', body=body)
```

Pada *script* di atas dapat dilihat bahwa pada saat proses *discovery network, control-plane* akan memanggil fungsi *self.MyDynRouteApp.discoveryNetwork* (dibahas pada sub bab Inisialisasi). Data kembalian dari fungsi tersebut akan dijadikan data kembalian dari modul ini dalam format JSON.

b. *applyFlowTableFromDb* (http://127.0.0.1:8080/*controller*/apply*FlowTable*FromDb/{*switch*Id})

Modul ini diakses oleh *application-plane* untuk memberikan triger kepada *control-plane* agar melakukan penerapan *flow-table* yang sudah tersimpan pada *database* tabel *t_flow_table*. Berikut adalah potongan *script* untuk modul *applyFlowTableFromDb* :

```
@route('controller',
MyDynRouteAppBaseUrl+'applyFlowTableFromDb/{switchId}')
def applyFlowTableFromDb(self, req, **kwargs):
      self.mLogger.info('wsgi '+str(req))
      dtReply = {}
      dtReply['status'] =
self.MyDynRouteApp.applyFlowTableFromDb(kwargs['switchId'])
      body =json.dumps(dtReply)
      return Response(content_type='application/json', body=body)
```

Dalam potongan *script* di atas, modul ini akan memanggil fungsi *self.MyDynRouteApp.applyFlowTableFromDb* (dibahas pada sub bab Penerapan *Flow-table*). Data kembalian dari fungsi tersebut akan dijadikan data kembalian dari modul ini dalam format json.

c. *emptyFlowTable* (http://127.0.0.1:8080/*controller*/empty*FlowTable*)

Modul ini diakses ketika diperlukan proses reset *flow-table* pada keadaan sistem pertama kali dijalankan. Modul ini dibuat untuk mempermudah melakukan pengujian pada penelitian. Berikut adalah potongan *script* untuk modul *emptyFlowTable* :

```
@route('controller',
MyDynRouteAppBaseUrl+'emptyFlowTable/{switchId}')
def emptyFlowTable(self, req, **kwargs):
      self.mLogger.info('wsgi '+str(req))
      dtReply = {}
      dtReply['status'] =
self.MyDynRouteApp.emptyFlowTable2(kwargs['switchId'])
      body =json.dumps(dtReply)
      return Response(content_type='application/json', body=body)
```

Dalam potongan *script* di atas, modul ini akan memanggil fungsi *self.MyDynRouteApp.emptyFlowTable2* (dibahas pada sub bab Penerapan *Flow-*

table). Data kembalian dari fungsi tersebut akan dijadikan data kembalian dari modul ini dalam format JSON.

3.2.4.3. Application-plane

Pada penelitian ini *application-plane* dipisahkan dengan *controller* dan keduanya berkomunikasi menggunakan protokol TCP *port number* 8080. Dalam berkomunikasi dengan *control-plane*, *application-plane* bertindak sebagai *socket-client*. *Application-plane* juga berkomunikasi dengan *database* untuk membaca dan menyimpan data. Komunikasi dengan *database* menggunakan *mysql-connector* yang berjalan dengan protokol TCP *port* 3306. *Application-plane* pada penelitian ini dibangun dengan menggunakan bahasa pemrograman *Python*. *Application-plane* juga berkomunikasi dengan sistem *monitoring sFlow-RT*. Komunikasi dengan *sFlow-RT* menggunakan protokol TCP *port number* 8008 dimana *application-plane* bertindak sebagai *socket-client*.

3.2.4.4. Sistem Monitoring

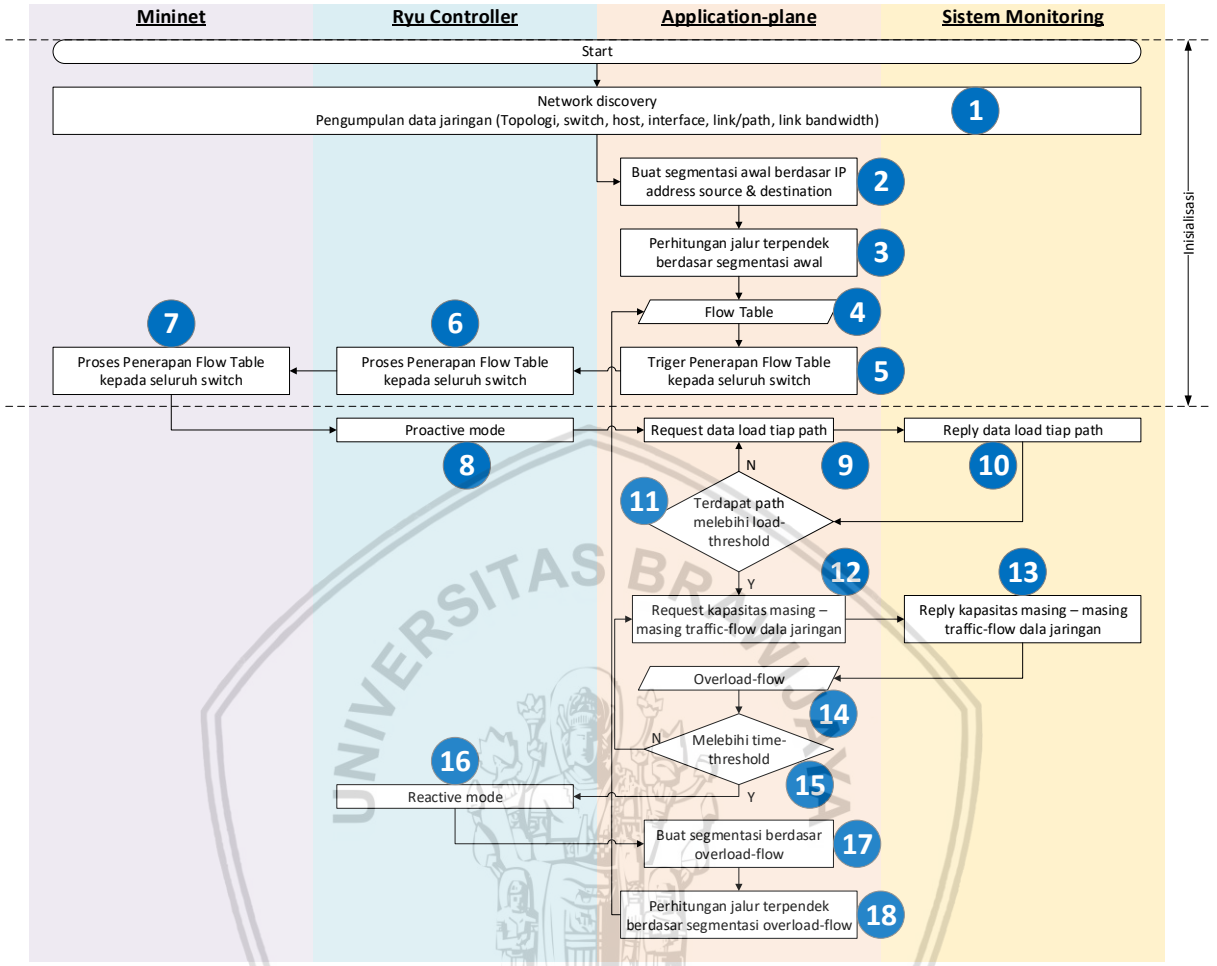
Sistem *monitoring* yang digunakan pada penelitian ini adalah *sFlow-RT* dan *bwm-ng*. *sFlow-RT* juga bertindak sebagai s*Flow-collector* yang berkomunikasi dengan *data-plane* menggunakan protokol UDP *port* 6343. Dalam berkomunikasi dengan *data-plane*, *sFlow-RT* bertindak sebagai *socket-server*. *sFlow-RT* juga melayani *request* melalui API (*Application Programming Interface*) yang disediakan dengan menggunakan protokol TCP *port* 8008. Dalam hal ini *sFlow-RT* bertindak sebagai *socket-server*. Komunikasi antar *application-plane* dan *sFlow-RT* menggunakan *sFlow-RT* API.

Dalam melakukan emulasi jaringan, *mininet* akan membentuk *virtual network interface* pada OS untuk setiap *network interface* yang diemulasikan. Untuk kepentingan penelitian dan untuk kemudahan proses, maka dalam penelitian ini digunakan *bwm-ng* untuk membaca beban *traffic-flow* pada setiap *virtual network interface*.

3.2.4.5. Database

Database yang digunakan pada penelitian ini adalah *Mysql*. *Mysql* menerima koneksi dari modul *control-plane* dan *application-plane*. Keduanya berkomunikasi melalui protokol TCP *port number* 3306. Dalam hal ini *Mysql* bertindak sebagai *socket-server*.

3.2.5. Cara Kerja Sistem



Gambar 3.5. Flowchart cara kerja sistem

Gambar 3.5 adalah *flowchart* cara kerja sistem secara keseluruhan. Berikut adalah keterangan untuk setiap sub proses pada *flowchart* di atas :

1. Sistem melakukan *network discovery*. Proses detail dalam *network discovery* dibahas pada sub bab berikutnya. Proses *network discovery* menghasilkan data :
   - *Switch* berdasar *dpid* (*datapath identity*).
   - *Host* berdasar MAC *address*.
   - MAC *address network interface switch* dan *host*
   - Koneksi (*link* / *path*) *switch-to-switch* dan *switch-to-host*
   - *Bandwidth path*
   - IP *address host*

   Dari data di atas dibentuk suatu permodelan *Dijkstra graph* untuk digunakan dalam proses perhitungan jalur terpendek.

2. Membuat daftar segmentasi *traffic-flow* berdasarkan IP *address* sumber dan tujuan dari seluruh kemungkinan *host-to-host*. Daftar IP *address* didapat dari proses poin 1. Proses segmentasi dan pemetaan *id* segmen dibahas pada sub bab berikutnya.
3. Menghitung jalur terpendek (menggunakan algoritma *Dijkstra*) dari setiap data segmen yang dihasilkan pada proses 2. Pada sub proses poin 3 ini nilai *weight*/*cost* awal yang diberikan pada setiap *path* adalah 1. Setiap iterasi perhitungan segmen, nilai *path* ditambahkan dengan 1 untuk setiap *path* yang dilewati segmen tersebut. Sub proses ini menghasilkan data pada poin 4.
4. Data *flow-table* adalah data *rule* yang akan diterapkan pada setiap *switch*. Data ini tersimpan pada tabel *t_flow_table* Ada dua bagian pada data ini, yaitu :
   - *Flow match*

     Flow match terdiri dari kolom :
     - *src_ipv4* : IP *address* sumber.
     - *dst_ipv4* : IP *address* tujuan.
     - proto : *internet protocol number* yang digunakan.
     - *src_proto_port* : *port number* yang digunakan pada sisi pengirim / sumber.
     - *dst_proto_port* : *port number* yang digunakan pada sisi penerima / tujuan.

     Hubungan logika pada daftar kolom di atas adalah AND dan akan diabaikan jika nilai dari kolom tersebut adalah '-1'.
   - *Action*

     Action berdasar kolom *out_port* yaitu nomor *index* dari *port* tertentu pada *switch* tersebut. *Port* tersebut digunakan untuk meneruskan setiap *traffic-flow* yang memenuhi persyaratan *flow match*.
5. *Application-plane* melakukan *trigger* kepada *controller* untuk melakukan penerapan data *flow-table* pada seluruh *switch* sesuai data pada poin 4. *Trigger* dilakukan dengan mengakses *web service* (WSGI) yang disediakan oleh *controller* dengan alamat URL http://127.0.0.1:8080/*controller*/apply*FlowTable*FromDb/All .
6. Berdasarkan data poin 4, *controller* akan mengirim *openflow message* kepada setiap *switch* pada kolom *switch_id* dengan atribut *flow match* dan *action*-nya.
7. *Switch* menerima *openflow message* dari *controller* dan menerapkan *flow-table* sesuai data *message* yang diterima pada *switch* tersebut.

8. Setelah semua *flow-table* diterapkan pada *switch*, maka *traffic-flow* yang masuk pada *switch* akan diproses dengan data *flow-table* internal pada *switch* tersebut. Dengan demikian *controller* akan bekerja dengan mode *proactive*.
9. *Application-plane* melakukan *monitoring* terhadap beban pada masing – masing *path*. *Monitoring* dengan melakukan perintah *bwm-ng* yang dilakukan dari *python application* melalui modul *shell*. Perintah yang dilakukan adalah `bwm-ng -o CSV -c 1`.
10. Modul *shell* mengeluarkan balikan berupa data dengan format CSV (seperti yang dibahas pada bab sebelumnya).
11. Dalam data CSV tersebut dapat dibaca nilai beban dari setiap *path* dengan membaca nilai jumlah *byte* yang keluar tiap detik pada masing – masing *network interface*. Dari data tersebut dideteksi apakah terdapat *path* dengan beban melebihi *load-threshold* (80% dari *bandwidth path*) dari *path* tersebut. Jika ada, maka dinyatakan sebagai *overload-path* dan menuju sub proses poin 12. Jika tidak ada maka kembali melakukan sub proses poin 9.
12. Melakukan *monitoring* seluruh *traffic-flow* yang terjadi pada jaringan. *Monitoring* dengan mengirimkan http *request* pada *sFlow-RT* dengan alamat URL http://127.0.0.1:8008/active*flows*/ALL/mn_*flow*/json?aggMode=avg.
13. *sFlow-RT* akan me-*reply* dengan data *traffic-flow* yang terjadi pada masing – masing *switch* beserta nilai kapasitas *traffic-flow* tersebut. Data yang dikirim menggunakan format JSON.
14. Data dari poin 15 ditambahkan data waktu pengambilan data dan disimpan dalam tabel *t_overload_flow* sebagai data *overload-flow*.
15. Jika *overload-path* terjadi melebihi batas waktu *time-threshold* maka menuju sub proses poin 16. Apabila tidak maka kembali pada sub proses 12.
16. *Controller* akan menghapus seluruh *flow-table* internal pada seluruh *switch*. *Controller* akan bekerja dengan mode *reactive* dan memberikan keputusan terhadap setiap *traffic-flow* pada *switch*. Keputusan yang diberikan oleh *controller* dihitung dengan menggunakan algoritma *Dijkstra*. Data yang dikrim pada *switch* adalah *action* (*out_port*) *traffic-flow* tersebut diteruskan.
17. Membuat daftar segmentasi *traffic-flow* berdasarkan data *overload-flow* yang dikumpulkan pada sub proses poin 12. Selain *dynamic port number* akan dinotasikan

dengan ‘-1’. Kemudian dilakukan perhitungan rata – rata terhadap *flow* yang sama. Dari daftar rata – data, *flow* dengan nilai rata – rata melebihi *minimum-segment-load* (dibahas pada sub bab berikutnya) akan diberikan *id* segmen tersendiri.

18. Menghitung jalur terpendek (menggunakan algoritma *Dijkstra*) dari setiap data segmen yang dihasilkan pada proses 17. Pada sub proses poin 18 ini nilai *weight*/*cost* awal yang diberikan pada setiap *path* adalah 1. Setiap iterasi perhitungan segmen, nilai *path* ditambahkan dengan nilai kapasitas rata – rata *traffic-flow* untuk setiap *path* yang dilewati segmen tersebut. Sub proses ini menghasilkan data *flow-table* dan selanjutnya kembali pada sub proses poin 4.

3.2.5.1. Inisialisasi

Pada tahap ini terdapat beberapa proses yang dilakukan. Berikut penjelasan proses yang dilakukan pada tahap inisialisasi :

a. *Network discovery*

Dalam proses ini dilakukan pembacaan data informasi mengenai jaringan. Data yang dibaca meliputi *switch*, *host*, *network interface*, *link* dan *bandwidth* yang digunakan tiap *link*. Data *switch* disimpan pada tabel *t_switch*. Data *host* disimpan pada tabel *t_host*. Data *network interface* disimpan pada tabel *t_intf*. Data *link* dan *bandwidth link* disimpan pada tabel *t_link*. Pada proses ini semua modul saling berkomunikasi dan saling bertukar data. Hasil dari proses ini digunakan dalam membentuk permodelan data *graph* untuk perhitungan menggunakan algoritma *Dijkstra*. Proses detail pada *network discovery* akan dibahas pada sub bab berikutnya.

b. Segmentasi awal

Setelah proses *network discovery* dilakukan, maka pada tabel *t_host* tersedia data semua *host* yang ada pada jaringan. Dari data IP *address* dari setiap *host*, dibentuk data segmentasi dengan protokol dan *port number wildcard*. *Wildcard* dilambangkan dengan simbol ‘-1’. Dengan demikian pada proses segmentasi awal ini hanya mempertimbangkan IP *address* sumber dan tujuan.

c. Perhitungan jalur terpendek

Berdasarkan data yang dihasilkan oleh proses segmentasi awal, selanjutnya dihitung menggunakan algoritma *Dijkstra* untuk menentukan jalur terpendek. Pada proses ini, seluruh *path* pada *Dijkstra graph* diberikan *weight* sama dengan 1. Proses perhitungan

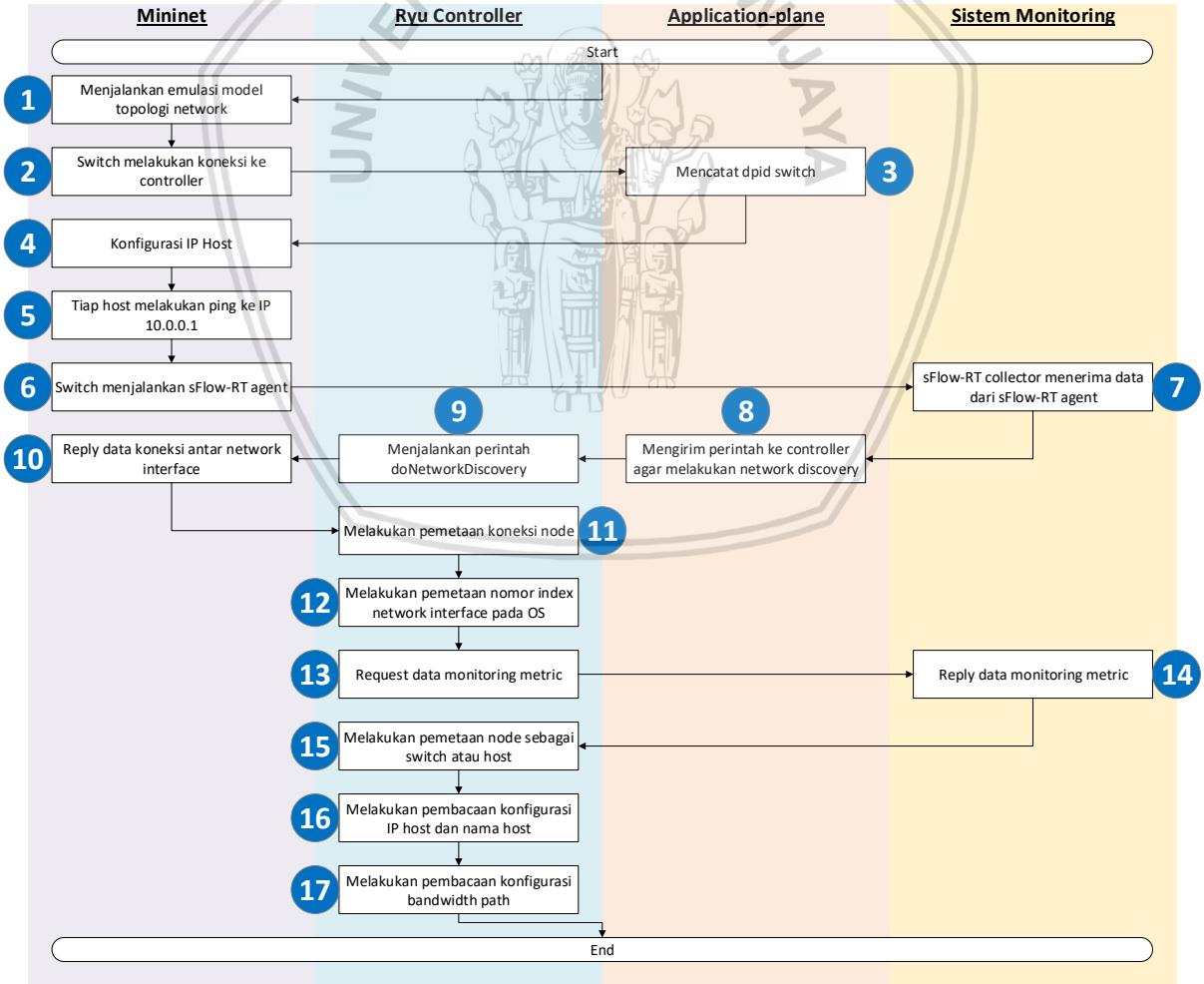
dilakukan terhadap setiap data segmen. Proses ini menghasilkan data *output* berupa *flow-table* yang tersimpan pada tabel *t_flow_table*.

d. Penerapan *flow-table*

Pada proses ini, data *flow-table* akan diterapkan terhadap seluruh *switch* pada jaringan. Proses ini dilakukan oleh *Ryu* berdasar perintah dari *application-plane* dengan data sesuai pada tabel *t_flow_table*. Urutan penerapan *flow-table* berdasar *hop_index* tertinggi. Setelah semua *flow-table* diterapkan, maka dengan demikian sistem akan bekerja secara *proactive*. Maka dengan demikian tahapan inisialisasi telah selesai dilakukan.

3.2.5.2. Network Discovery

Proses *network discovery* melibatkan seluruh modul pada sistem. Berikut adalah *flowchart* dari proses *network discovery* :



Gambar 3.6. Flowchart network discovery

Gambar 3.6 adalah *flowchart* cara kerja sistem secara keseluruhan. Berikut adalah keterangan untuk setiap sub proses pada *flowchart* di atas :

1. Menjalankan *script python* (*script* dengan nama file fat-tree-A.py terlampir). Di dalam *script* ini menggunakan *mininet* API untuk menjalankan fungsi – fungsi yang terdapat dalam *mininet*. Beberapa fungsi yang dilakukan adalah membentuk *virtual switch* dan *host* serta membuat koneksi *switch-to-switch* dan *switch-to-host*.
2. Dalam *script* poin 1 juga dijalankan perintah mengkoneksikan *switch* dengan *controller*.
3. *Controller* mencatat *dpid* dari setiap *switch* yang terkoneksi.
4. Dalam *script* poin 1 juga dijalankan perintah untuk mengkonfigurasi IP *address* dari setiap *host*.
5. Dalam *script* poin 1 juga dilakukan perintah untuk melakukan *ping* dari masing – masing *host* ke IP 10.0.0.1.
6. Dalam *script* poin 1 juga dijalankan perintah untuk menjalankan s*Flow* agent pada masing – masing *switch*.
7. Sistem *monitoring sFlow-RT* menerima data informasi tentang *switch* dan memetakan menjadi *monitoring metric*.
8. *Application-plane* memberikan *trigger* ke *controller* untuk melakukan proses *network discovery* dengan melakukan HTTP *request* ke *controller* dengan alamat URL http://127.0.0.1:8080/controller/discoveryNetwork.
9. *Controller* melakukan *request* data koneksi *network interface*. Dari data ini didapat MAC *address* yang terkoneksi terhadap masing – masing *network interface* pada *switch*.
10. *Switch* mengirimkan data MAC *address* dan IP *address network interface* yang terkoneksi pada masing – masing *switch*.
11. *Controller* melakukan pemetaan antara koneksi antar MAC *address*.
12. Melakukan pemetaan nomor *index network interface* berdasar OS. *Index* OS *interface* digunakan untuk memetakan terhadap data *monitoring*.
13. *Request* data *monitoring metric* dengan mengirimkan HTTP *request* pada *sFlow-RT* dengan alamat URL http://127.0.0.1:8008/metric/127.0.0.1/json.
14. *sFlow-RT* mengirimkan data *monitoring metric*. Data yang dikirim adalah nomor *index interface* pada *switch* berdasar MAC *address*.

15. Berdasar data poin 10 dan 14 maka dapat dipetakan MAC *address switch* dan MAC *address host*.
16. Melakukan pembacaan nama *host* berdasar IP *address* data poin 10 pada *script* yang digunakan pada poin 1.
17. Melakukan pembacaan konfigurasi *bandwidth path* bedasar nama *host* dan nama *switch* pada *script* yang digunakan pada poin 1.

Dengan demikian sistem mempunyai data *switch*, *host*, *network interface*, IP *address* MAC *address*, koneksi *switch-to-switch*, koneksi *switch-to-host* dan *bandwidth* dari masing – masing koneksi. Data *host* disimpan dalam tabel *t_host*, data *switch* disimpan pada tabel *t_switch*, data *network interface* disimpan pada tabel *t_intf* dan data koneksi disimpan pada tabel *t_link*.

3.2.5.3. Memuat Data *Dijkstra Graph*

Permodelan data *Dijkstra graph* berdasarkan hasil data pada proses *network discovery*. Data *switch* dan *host* digunakan sebagai *node* pada data *graph*. Berdasar data pada tabel *t_link* dapat dibentuk *edge* / *path* antar *node*. Setiap koneksi antar *node* dibuatkan dua *edge* dengan arah yang saling berlawanan. Diperlukan dua *edge* dikarenakan jalur pengiriman dan penerimaan terpisah. Dengan demikian data *graph* siap untuk dilakukan perhitungan menggunakan algoritma *Dijkstra*.

3.2.5.4. Segmentasi

Segmentasi adalah proses pengelompokan *traffic-flow*. Proses segmentasi dilakukan pada tahap inisialisasi dan ketika proses perhitungan ulang *route-path*. Segmentasi pada tahap inisialisasi hanya mempertimbangkan IP *address* sumber dan tujuan untuk membedakan segmen. Contoh *flow* dari h1 dengan IP *address* 10.0.2.1 dengan tujuan h15 dengan IP *address* 10.0.2.15, maka *id* segmennya adalah ’10.0.2.1/-1/-1/-1/10.0.2.15’ dimana ‘-1’ adalah notasi untuk seluruh protokol dan seluruh *port number*. Blok pemetaan *id* segmen seperti pada gambar 3.7 berikut :

IP address sumber	/	Port number sumber	/	Internet protocol number	/	Port number tujuan	/	IP address tujuan

Gambar 3.7. Pemetaan id segmen traffic-flow

Sedangkan segmentasi pada saat perhitungan ulang *route-path* adalah berdasar data *traffic-flow* yang terjadi saat adanya *overload-path* dalam rentang *time-threshold*. Segmentasi pada tahap ini mempertimbangkan IP *address* sumber dan tujuan, *internet protocol number* serta *port number* sumber dan tujuan. Setiap *traffic-flow* dengan kapasitas melebihi *minimum-segment-load* maka dibuatkan ID segmen tersendiri. Sedangkan *traffic-flow* yang tidak memenuhi *minimum-segment-load* tidak dibuatkan segmen ID, melainkan digabung dalam segmen yang hanya mempertimbangkan IP *address* sumber dan tujuan seperti halnya pada tahap inisialisasi. Segmen ini dinamakan *wildcard-segment*. Dimana *wildcard-segment* ini diberikan prioritas lebih rendah dalam *flow-table* daripada *non-wildcard-segment*.

Berikut adalah cuplikan *source code* untuk proses segmentasi awal :

```
def generateDefaultSegment(self, isEmptyExisting = True):
      self.mLogger.info('generate default segment')
      self.mGraph = self.loadGraph()
      CSVFile = self.mCurrentDir+'/tmp/defaultSegmen.CSV'
      fileHdl = open(CSVFile, 'w')
      if isEmptyExisting:
            self.mDatabase.execQuery('delete from `t_traffic_segment`')
      pGraph = self.loadGraph()
      for pNode1 in pGraph.nodes():
            for pNode2 in pGraph.nodes():
                  if (
                        (pNode1!=pNode2) and
                        (pGraph.node[pNode1]['data']['type'] == 'HOST')
and
                        (pGraph.node[pNode2]['data']['type'] == 'HOST')
                  ):
                        srcIp = pGraph.node[pNode1]['data']['ipv4']
                        dstIp = pGraph.node[pNode2]['data']['ipv4']
                        segmentId = srcIp+'/-1/-1/-1/'+dstIp
                        CSVLine = segmentId+"\t"+ \
                                    srcIp+"\t"+ \
                                    '-1'+"\t"+ \
                                    '-1'+"\t"+ \
                                    '-1'+"\t"+ \
                                    dstIp+"\t"+ \
                                    '1'+"\n"
                        fileHdl.write(CSVLine)
      fileHdl.close()
      strSql = 'load data local infile "'+CSVFile+'" into table
`t_traffic_segment` ('+ \

      '`id`,`src_ipv4`,`src_proto_port`,`proto`,`dst_proto_port`,`dst_ipv
4`,`load`'+ \
                        ') '
      self.mDatabase.execQuery(strSql)
      return 'SUCCESS'
```

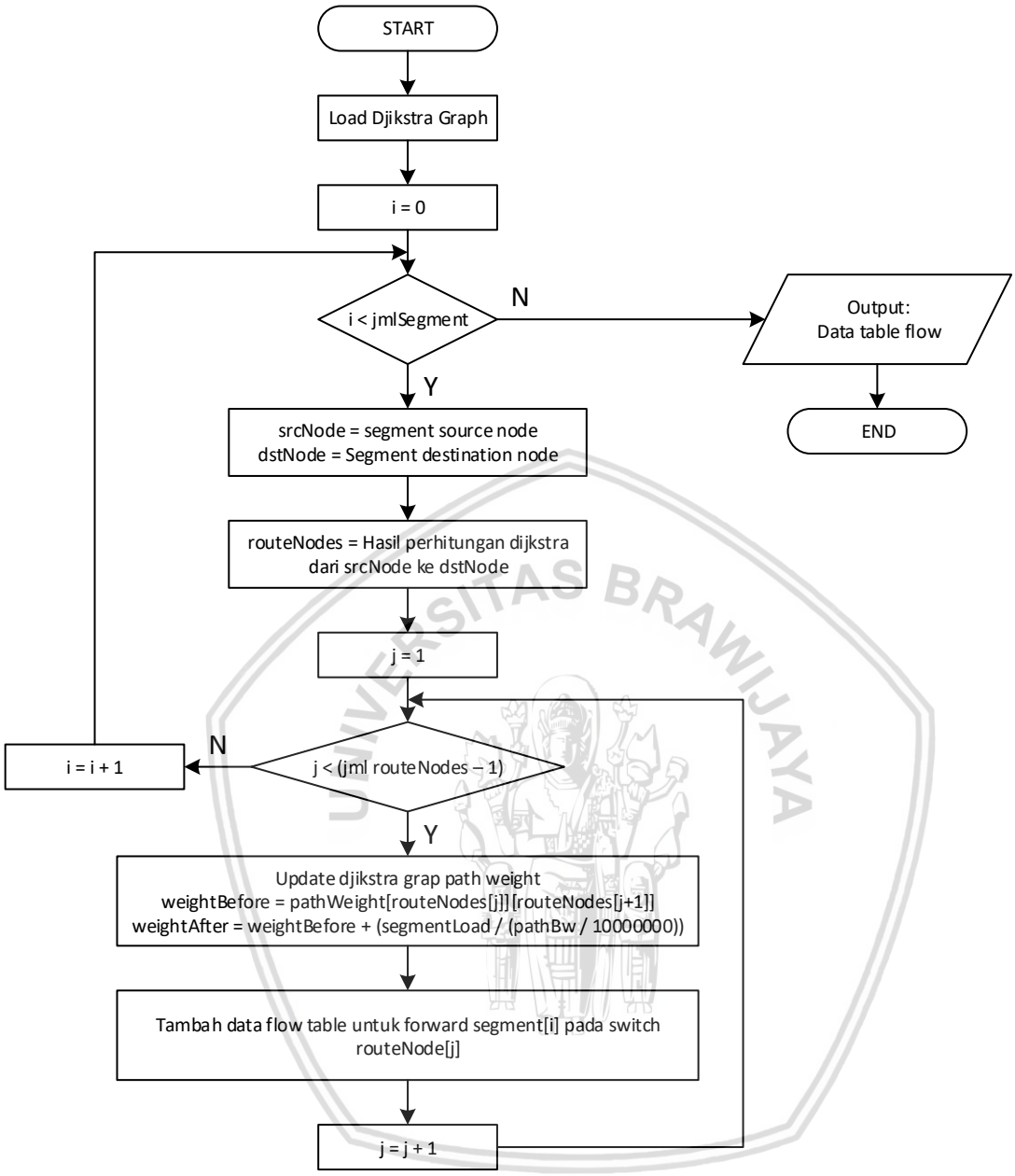
3.2.5.5. Perhitungan *Route-Path*

Data dasar yang digunakan dalam proses ini adalah data segmen *traffic-flow* hasil dari proses inisialisasi atau proses segmentasi setelah terjadi *overload-path* melebihi batas *time-threshold*. Data segmen tersimpan dalam tabel *t_traffic_segment* dalam *database*. Nilai bobot *path* atau *path-cost* diformulasikan sebagai berikut :

$$w_{(a,b)} = \frac{L_{(a,b)}}{(BW_{(a,b)}/k)} \quad \text{.....(1)}$$

Dimana :

- *L* adalah *load-path* atau beban *traffic-flow* pada *path* tersebut
- *k* adalah konstanta acuan nilai *bandwidth*, k = 10000000.
- *w* adalah nilai *path-cost*
- *a* adalah *node source (host/switch source)*
- *b* adalah *node destination (switch destination)*
- *BW* adalah nilai *bandwidth path*.



Gambar 3.8. Flowchart perhitungan route-path traffic-flow

Gambar 3.8 adalah *flowchart* untuk proses perhitungan *route-path* dari setiap segmen *traffic-flow*. Pada proses perhitungan sistem akan membaca data segmen *traffic-flow* yang terdapat pada tabel *t_traffic_segment*. Setiap segmen akan dilakukan perhitungan menggunakan *Dijkstra* berdasar model *graph* yang telah dibentuk pada saat tahap inisialisasi. Setiap perhitungan segmen, sistem akan memperbarui nilai *cost*/*weight* pada *edge-graph Dijkstra* untuk proses perhitungan segmen berikutnya. Setiap perhitungan segmen akan menghasilkan data sederetan *route-path* untuk segmen tersebut. Tiap data

route-path akan diterjemahkan menjadi data *flow-table* dan disimpan dalam tabel *t_flow_table*.

3.2.5.6. Penerapan *Flow-table*

Pada proses ini data pada tabel *t_flow_table* dibaca berdasarkan *switch* dan diurutkan berdasar *hop_index* terbesar. Setiap data *flow-table* akan diterapkan menggunakan *openflow message* oleh *Ryu* kepada *switch* dari data tersebut.

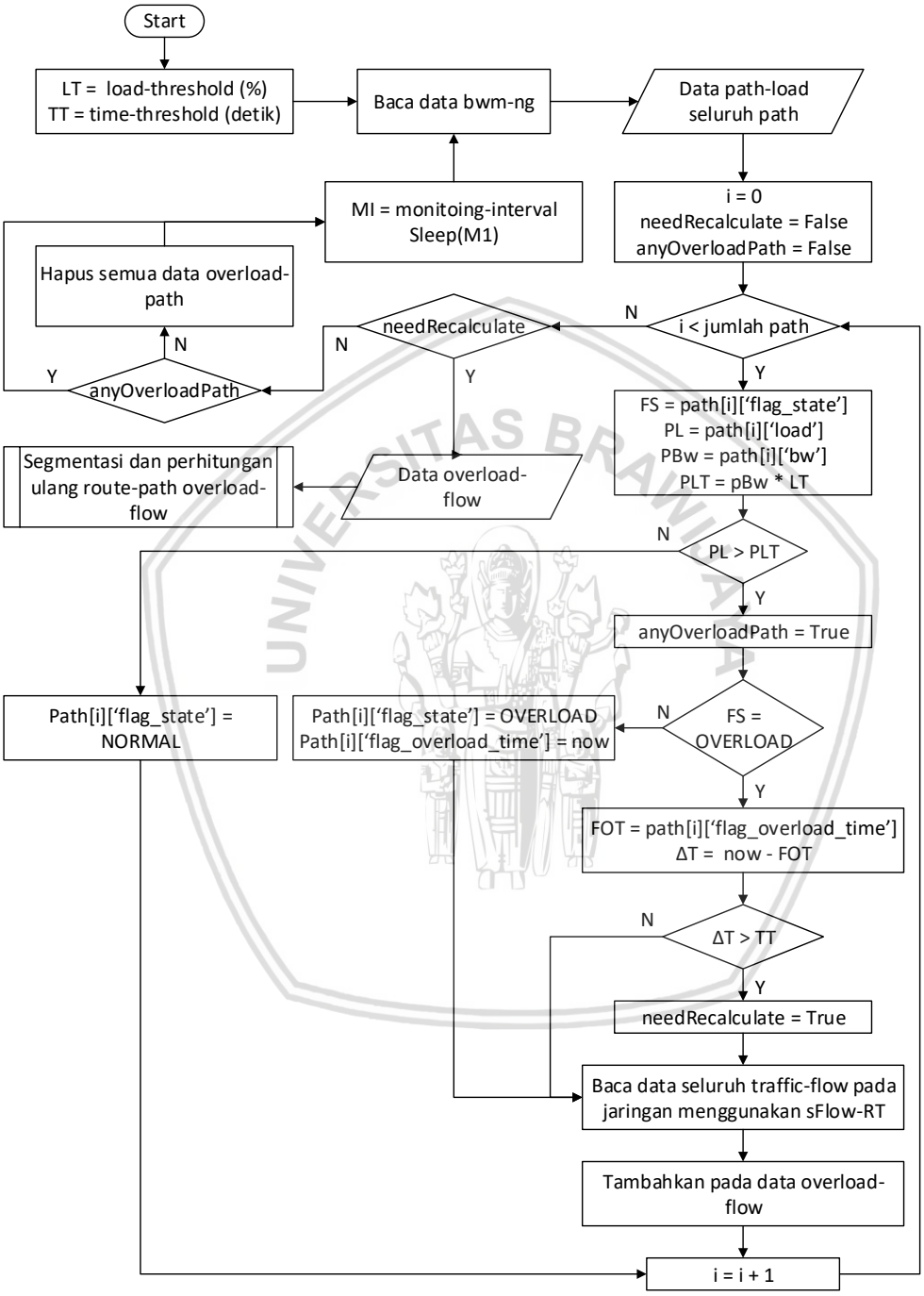
3.2.5.7. Monitoring

Monitoring dilakukan tiap beberapa detik. Variabel *monitoring-interval* adalah *interval* waktu untuk melakukan *monitoring*. *Monitoring* dilakukan ketika sistem berjalan dengan mode *proactive*. *Monitoring* yang dilakukan menggunakan *bwm-ng* untuk mendapatkan data total beban *traffic-load* pada setiap *network interface*. Data ini mewakili total beban *traffic-load* pada *path* yang terhubung dengan *network interface* tersebut. Data ini disebut dengan *path-load*. Data *path-load* disimpan dalam tabel *t_intf*.

Selain *bwm-ng*, *monitoring* juga menggunakan *sFlow-RT* untuk membaca semua *traffic-flow* yang terjadi pada jaringan. *Monitoring* menggunakan *sFlow-RT* dilakukan dalam rentang waktu *time-threshold*. Data *traffic-flow* ketika terjadi *overload-path* selama *time-threshold* disebut sebagai data *overload-flow*. Data *overload-flow* disimpan pada tabel *t_overload_flow*. Data ini akan menjadi data dasar untuk proses segmentasi.

3.2.5.8. Deteksi *Overload*

Proses deteksi *overload* berjalan bersamaan dengan proses *monitoring*. Hubungan proses *monitoring* dan deteksi *overload* digambarkan dalam *flowchart* sebagai berikut :



Gambar 3.9. Flowchart monitoring dan deteksi overload

Pada gambar 3.9 nampak hubungan antara proses *monitoring* dengan proses deteksi *overload*. Dimana tiap *monitoring-interval* sistem melakukan pembacaan data *monitoring bwm-ng* untuk mendeteksi apakah terjadi *overload-path*. Pada saat terjadi *overload-path*

maka sistem akan mencatat waktu awal mula terjadi *overload* pada *path* tersebut (*path-flag-overload-time*). Jika ada *overload-path* maka sistem akan melakukan pembacaan data *overload-flow*, jika tidak maka sistem akan menghapus seluruh data *overload-flow*. Demikian secara terus menerus setiap rentang waktu *monitoring-interval* sistem melakukan hal tersebut. Selain itu sistem juga akan mendeteksi apakah ada data *overload-path* yang telah melebihi *time-threshold*. Jika ada maka sistem akan memutuskan untuk melakukan perhitungan ulang *route-path* bagi seluruh *overload-flow*.

3.2.6. Simulasi

Simulasi jaringan dan *data-plane* menggunakan *mininet* dengan *controller* menggunakan *Ryu*. *Switch* yang digunakan adalah *open-vSwitch* dengan menggunakan *protocol openflow*. *Application-plane* dibangun dengan menggunakan bahasa pemrograman *Python*. Sedangkan *database* menggunakan *database Mysql*.

3.2.7. Pengujian

Pada penelitian ini dilakukan tiga macam pengujian, yaitu

a. Pengujian *throughput*

Pada pengujian ini ditujukan untuk menguji kemampuan sistem dalam meratakan beberapa *traffic-flow* agar tidak terjadi *overload-path* sehingga perolehan *bandwidth throughput traffic-flow* secara keseluruhan dapat optimal.

Pada pengujian ini dilakukan pengiriman beberapa *traffic-flow* secara bersamaan. *Traffic-flow* yang dikirimkan seperti pada tabel 3.1.

Tabel 3.18. Tabel traffic-flow pengujian throughput

ID	*Source Node*	*Destination Node*	*Protocol*	*Port Number*	*Load (Mbps)*	*Time* (minutes)
FL1	h1	h16	UDP	3001	5	10
FL2	h2	h17	UDP	3002	5	10
FL3	h3	h18	UDP	3003	5	10
FL4	h4	h19	UDP	3004	5	10
FL5	h5	h20	TCP	2001	2.5	10
FL6	h5	h20	TCP	2002	2.5	10
FL7	h5	h20	TCP	2003	2.5	10
FL8	h5	h20	TCP	2004	2.5	10

Tabel 3.18 adalah tabel spesifikasi *traffic-flow* yang dikirimkan secara bersamaan pada pengujian ini. *Traffic-flow* yang dikirimkan sebanyak 8 *traffic-flow* dengan 4 *traffic-flow* menggunakan protokol TCP dan 4 *traffic-flow* menggunakan UDP, masing – masing dikirimkan selama waktu 10 menit. Parameter yang diukur pada pengujian ini adalah total perolehan *throughput* dari seluruh *traffic-flow* yang dikirimkan.

b. Pengujian latensi

Pengujian latensi *flow* dilakukan untuk mengukur kecepatan pengiriman *traffic-flow* dari sumber sampai dengan tujuan. Skenario pada pengujian ini adalah pengiriman *traffic-flow* pada pengujian *throughput* tetap dilakukan dan ditambah proses *copy* file menggunakan perintah *scp*. Ukuran file yang di-*copy* adalah sebesar 50 Mb. Pengiriman file dilakukan dari h1 ke h20.

c. Pengujian respon sistem

Pengujian ini dilakukan untuk melihat respon sistem ketika terjadi lonjakan trafik sesaat (*traffic*-spike). Ketika terjadi *traffic*-spike seharusnya sistem tidak perlu melakukan pengalihan trafik dikarenakan sifat dari *traffic*-spike yang hanya terjadi dalam rentang waktu yang sangat singkat. Hal ini perlu diperhatikan mengingat pengalihan trafik akan berdampak pada penurunan *throughput flow* secara keseluruhan. Dikarenakan pada pengujian ini berdampak juga pada nilai *throughput flow*, maka pengujian ini akan dilakukan hanya pada metode dengan hasil pengujian *throughput* terbaik dan hasil grafik pemerataan paling baik. Jika pada pengujian *throughput* sebelumnya hanya diberikan beban 8 *traffic-flow* statis, maka pada pengujian respon sistem akan diberikan beban *traffic-flow* secara berkala selama 5 detik sebanyak 5 kali. Skenario tersebut untuk mensimulasikan terjadinya *traffic*-spike secara berulang pada jaringan. Spesifikasi *traffic-flow* yang diberikan untuk mensimulasikan *traffic*-spike seperti yang ditampilkan pada tabel 3.19.

Tabel 3.19. Tabel traffic-flow pengujian respon sistem

ID	*Source Node*	*Destination Node*	*Protocol*	*Port Number*	*Load*	*Time* (detik)
FL5	h6	h11	UDP	3001	9 Mbps	5

Pada tabel 3.19 nampak *traffic-flow* yang dikirimkan pada pengujian respon sistem sebanyak satu *traffic-flow*. *Traffic-flow* tersebut diberikan nilai 9 Mbps dan dikirimkan selama 5 detik sebanyak 5 kali. *Traffic-load* tersebut dapat mewakili terjadinya *traffic-spike* dalam jaringan. Parameter yang diukur adalah apakah sistem melakukan perhitungan ulang terhadap *traffic-spike* atau mampu menahan dan mendeteksi apakah kenaikan beban *path-load* hanya karena *traffic-spike* sesaat.

BAB IV
METODOLOGI PENELITIAN

4.1. Variabel dan Konstanta

Berikut beberapa variabel dan konstanta yang digunakan metode Hybrid dalam penelitian ini :

- *Bandwidth path* = 10 Mbps
- *Load-threshlod* = 80% dari *bandwidth path* = 8 Mbps
- *Time-threshold* = 15 detik

4.2. Langkah Menjalankan Sistem

Berikut adalah langkah untuk menjalankan sistem :

1. Menjalankan *mininet* melalui *mininet* API yang ditulis menggunakan bahasa pemrograman *Python*. *Source code* terlampir dengan mana file *fat-tree*-A.py. Cara menjalankan dengan menjalankan *shell command* '*python fat-tree*-A.py' pada *shell* OS.
2. Menjalankan *Ryu controller*. Perintah untuk menjalankan *controller* adalah sebagai berikut :
   a. Slavica ➔ *ryu-manager slavica-ryu.py --observe-links*
   b. LABERIO ➔ *ryu-manager laberio-ryu.py --observe-links*
   c. DLPO ➔ *ryu-manager dlpo-ryu.py --observe-links*
   d. Hybrid ➔ *ryu-manager dsi-ryu.py --observe-links*
3. Menjalankan proses inisialisasi
   Perintah untuk menjalankan proses inisialisasi adalah sebagai berikut :
   a. Slavica ➔ *./slavica-app.py doPreparationProcess*
   b. LABERIO ➔ *./laberio-app.py doPreparationProcess*
   c. DLPO ➔ *./dlpo-app.py doPreparationProcess*
   d. Hybrid ➔ *./dsi-app.py doPreparationProcess*
4. Menjalankan *application-plane*
   Perintah untuk menjalankan *application-plane* adalah sebagai berikut :
   a. Slavica ➔ *./slavica-app.py doMonitoringPathLoad*

b. LABERIO ➔ *./laberio-app.py doMonitoringPathLoad*
c. DLPO ➔ *./dlpo-app.py doMonitoringPathLoad*
d. Hybrid ➔ *./dsi-app.py doMonitoringPathLoad*

4.3. Langkah Pengujian

Setelah proses inisialisasi dan menjalankan sistem, maka sistem siap untuk dilakukan pengujian. Pada proses pengujian menggunakan *command* 'm' pada *mininet* yang berfungsi untuk memberikan perintah kepada *host* yang diemulasikan oleh *mininet*. Berikut akan dijelaskan langkah – langkah dalam melakukan pengujian sistem.

4.3.1. Pengujian *Throughput*

Pada pengujian ini dijalankan beberapa *shell command* secara bersamaan. Aplikasi yang digunakan dalam pengujian ini adalah *iperf* dan *iperf*3. *Iperf* digunakan untuk *traffic-flow* dengan protokol UDP dan *iperf*3 digunakan untuk *traffic-flow* dengan protokol TCP. Penggunaan *iperf*3 untuk protokol TCP dikarenakan fitur ukuran *flow* yang dikirim untuk *flow* TCP yang tidak terdapat pada *iperf*. Daftar *shell command* yang dilakukan adalah sebagai berikut :

Tabel 4.1. Tabel chell command pengujian throughput

ID *Flow*	***Shell Command***
FL1	a. ./m h16 *iperf* –s –u –p 3001 b. ./m h1 *iperf* –c 10.0.2.16 –u –p 3001 –b 5M –i 1 –t 600
FL2	a. ./m h17 *iperf* –s –u –p 3002 b. ./m h2 *iperf* –c 10.0.2.16 –u –p 3002 –b 5M –i 1 –t 600
FL3	a. ./m h18 *iperf* –s –u –p 3003 b. ./m h3 *iperf* –c 10.0.2.16 –u –p 3003 –b 5M –i 1 –t 600
FL4	a. ./m h19 *iperf* –s –u –p 3004 b. ./m h4 *iperf* –c 10.0.2.16 –u –p 3004 –b 5M –i 1 –t 600
FL5	a. ./m h20 *iperf*3 –s –p 2001 b. ./m h1 *iperf*3 –c 10.0.2.16 –p 2001 –b 2500K –i 1 –t 600
FL6	a. ./m h20 *iperf*3 –s –p 2002 b. ./m h1 *iperf*3 –c 10.0.2.16 –p 2002 –b 2500K –i 1 –t 600
FL7	a. ./m h20 *iperf*3 –s –p 2003 b. ./m h1 *iperf*3 –c 10.0.2.16 –p 2003 –b 2500K –i 1 –t 600
FL8	a. ./m h20 *iperf*3 –s –p 2004 b. ./m h1 *iperf*3 –c 10.0.2.16 –p 2004 –b 2500K –i 1 –t 600

Urutan menjalankan *shell command* pada tabel 4.1 adalah sebagai berikut :

- Menjalankan *shell command* poin a untuk FL1, FL2, FL3, FL4, FL5, FL6, FL7 dan FL8.
- Menjalankan *shell command* point b untuk FL1, FL2, FL3, FL4, FL5, FL6, FL7 dan FL8 secara bersamaan.

Hasil yang diambil adalah bagian terakhir *output shell command* pengujian *throughput*. Berikut contoh *output shell command* pengujian *throughput* :

```
[  5] Server Report:
[  5]  0.0-600.0 sec   355 MBytes  4.96 Mbits/sec   0.514 ms 2208/255102 (0.87%)
```

Gambar 4.1. Contoh output perintah iperf

Pada gambar 4.1 nampak contoh *output* dari perintah *iperf* yang digunakan pada *shell command* pengujian *throughput*. Pada contoh diatas, nilai yang diambil adalah 4,96 Mbps yang menandakan perolehan *throughput* dari *traffic-flow* tersebut.

4.3.2. Pengujian Latensi

Pada pengujian ini dilakukan prosedur seperti pada pengujian *throughput* dan ditambahkan proses *copy* file berukuran 50 MB dengan nama ‘file.50MB’ dari h5 ke h20.

```
/mininet/util# date && ./m h5 scp file.50MB root@10.0.2.20:/var/tmp/file.50MB && date
```

Gambar 4.2. Shell command copy file via scp

Gambar 4.2 adalah *shell command* untuk proses *copy* file dari h5 ke h20 menggunakan perintah *scp*. Perintah date pada awal dan akhir *shell command* untuk mendapatkan waktu awal proses *copy* file dilakukan dan waktu berakhir proses *copy*. Dari selisih kedua waktu tersebut dapat disimpulkan sebagai lama proses *copy* file dilakukan. Lama proses *copy* file inilah nilai yang diambil dalam pengujian ini.

4.3.3. Pengujian Respon Sistem

Shell command yang digunakan pada pengujian ini adalah *‘./m h6 iperf –c 10.0.2.11 –u –p 3001 –b 5M –i 1 –t 5*’. *Shell command* tersebut dijalankan sebanyak 5 kali dalam rentang waktu pengujian *throughput* (10 menit).

BAB V
HASIL DAN PEMBAHASAN

5.1. Pengujian *Throughput*

Dengan mengacu pada konsep dan tujuan pengujian *throughput* pada bab 3 dan mengikuti langkah – langkah teknis pengujian pada bab 4 maka pada pengujian *throughput* diperoleh hasil pengujian yang ditampilkan pada tabel 5.1.

Tabel 5.1. Segmentasi *traffic-flow pada pengujian throughput*

ID	Src *Node*	Dst *Node*	*Protocol*	*Port* Num.	Segmentasi			
					Slavica	LABERIO	DLPO	Hybrid
FL1	h1	h16	UDP	3001	SlvSegUDP1	LbrSegUDP1	DlpSegUDP1	HbrSegUDP1
FL2	h2	h17	UDP	3002	SlvSegUDP2	LbrSegUDP2	DlpSegUDP2	HbrSegUDP2
FL3	h3	h18	UDP	3003	SlvSegUDP3	LbrSegUDP3	DlpSegUDP3	HbrSegUDP3
FL4	h4	h19	UDP	3004	SlvSegUDP4	LbrSegUDP4	DlpSegUDP4	HbrSegUDP4
FL5	h5	h20	TCP	2001	SlvSegTCP1	LbrSegTCP1	DlpSegTCP1	HbrSegTCP1
FL6	h5	h20	TCP	2002				HbrSegTCP2
FL7	h5	h20	TCP	2003				HbrSegTCP3
FL8	h5	h20	TCP	2004				HbrSegTCP4

Tabel 5.1 menunjukkan segmentasi dari *traffic-flow* pada setiap metode. Dalam melakukan perhitungan *route-path*, masing – masing metode akan merujuk pada data segmentasi *traffic-flow*. Tiap segmen *traffic-flow* akan melewati *route-path* yang sama.

Tabel 5.2. Hasil throughput traffic-flow pada pengujian throughput

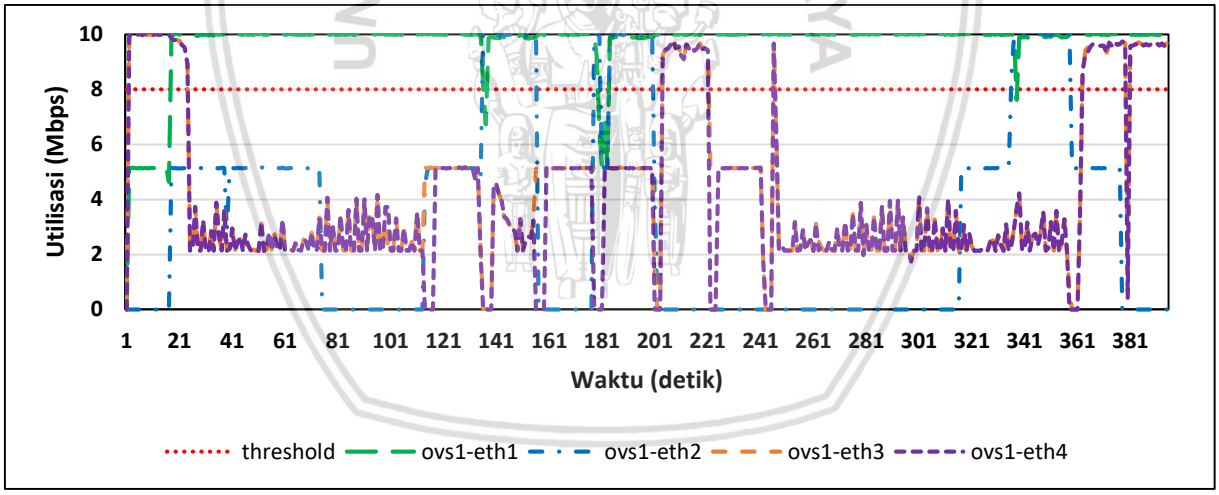
ID	*Src Node*	*Dst Node*	*Protocol*	*Port Num.*	*Load (Mbps)*	*Time (minutes)*	*Bandwidth (Mbps)*			
							Slavica	**LABERIO**	**DLPO**	**Hybrid**
FL1	h1	h16	UDP	3001	5	10	3.53	4.84	4.97	4.95
FL2	h2	h17	UDP	3002	5	10	4.31	5	5	4.84
FL3	h3	h18	UDP	3003	5	10	4.5	4.79	5	4.8
FL4	h4	h19	UDP	3004	5	10	3.74	5	5	4.94
FL5	h5	h20	TCP	2001	2.5	10	1.17	1.55	1.33	2.28
FL6	h5	h20	TCP	2002	2.5	10	2.29	1.65	1.12	2.23
FL7	h5	h20	TCP	2003	2.5	10	0.352	1.76	1.38	2.18
FL8	h5	h20	TCP	2004	2.5	10	1.03	1.51	1.24	2.26
UDP			**Total**				**16.08**	**19.63**	**19.97**	**19.53**
			Difference with Hybrid				**3.45**	**-0.1**	**-0.44**	**0**
TCP			**Total**				**4.842**	**6.47**	**5.07**	**8.95**
			Difference with Hybrid				**4.108**	**2.48**	**3.88**	**0**
UDP & TCP			**Total**				**20.922**	**26.1**	**25.04**	**28.48**
			Difference with Hybrid				**7.558**	**2.38**	**3.44**	**0**

Tabel 5.2 adalah tabel hasil perolehan *throughput* tiap *traffic-flow*. Pada *traffic-flow* UDP terlihat metode Hybrid memperoleh *throughput* lebih kecil 0,1 Mbps dari LABERIO dan 0,44 Mbps dari metode DLPO. Hal ini disebabkan karena pada metode LABERIO dan DLPO hanya melakukan perhitungan ulang terhadap *traffic-flow* yang terdapat pada *overload-path,* sedangkan metode Hybrid melakukan perhitungan ulang terhadap *traffic-flow* yang terdapat pada seluruh *path*. Ketika perhitungan ulang tersebut maka terjadi penurunan *throughput* sesaat untuk keseluruhan *traffic-flow*. Pada metode LABERIO dan DLPO sangat menguntungkan bagi *traffic-flow* FL1, FL2 FL3 dan FL4 dikarenakan berada dalam segmen yang berbeda dan karena *path* egress dari ovs1 hanya 4, maka salah satu *path* akan terdapat *traffic-flow* UDP dan TCP secara bersamaan. Jika *traffic-flow* UDP dan TCP terdapat pada *path* yang sama, maka berdampak pada perolehan *throughput traffic-flow* TCP seperti yang terlihat pada hasil *throughput* TCP pada metode LABERIO dan DLPO. Sedangkan jika dibandingkan dengan metode Slavica untuk perolehan *traffic-flow* UDP, metode Hybrid memperoleh *throughput* lebih besar 3,45 Mbps.

Untuk *traffic-flow* TCP, metode Hybrid memperoleh *throughput* lebih besar 4,108 Mbps dari Slavica, 2,48 Mbps dari LABERIO dan 3,88 Mbps dari DLPO. Sama halnya pada

UDP, metode Hybrid akan memisahkan *traffic-flow* dengan *port number* berbeda ke dalam segmen yang berbeda sehingga FL5, FL6, FL7 dan FL8 dapat memperoleh *route-path* yang berbeda. Jika melihat pada data segmentasi *traffic-flow*, maka pada metode Hybrid, masing – masing *traffic-flow* UDP akan mendapat *route-path* yang sama dengan salah satu *traffic-flow* TCP. Jika salah satu *traffic-flow* UDP dan salah satu *traffic-flow* TCP terdapat pada *path* yang sama, jika dilihat dari total kapasitas yang dikirimkan (5 Mbps + 2.5 Mbps = 7.5 Mbps), maka masih berada dibawah batas *load-threshold* (8 Mbps). Karena total *traffic-load* masih di bawah nilai *load-threshold* maka *traffic-flow* UDP tidak terlalu mempengaruhi *traffic-flow* TCP. Hal ini menyebabkan perolehan *throughput traffic-flow* secara keseluruhan pada metode Hybrid lebih besar dari pada metode lainnya.

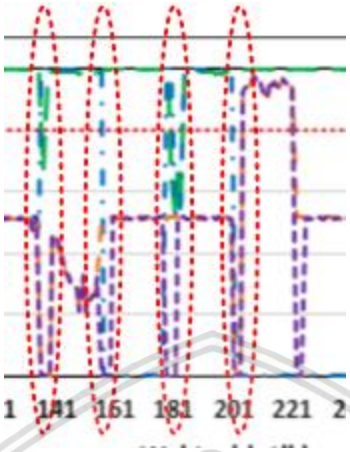
Untuk perolehan *throughput traffic-flow* secara keseluruhan, metode Hybrid memperoleh *throughput* lebih besar 7,558 Mbps dari Slavica, 2,38 Mbps dari LABERIO dan 3,44 Mbps dari DLPO. Meskipun pada *traffic-load* UDP lebih kecil dari pada LABERIO dan DLPO, akan tetapi pada TCP masih lebih besar dari metode lainnya. Sehingga perolehan *throughput traffic-flow* secara keseluruhan dapat lebih besar dari pada metode lainnya.



Gambar 5.1. Grafik path-load dengan metode Slavica

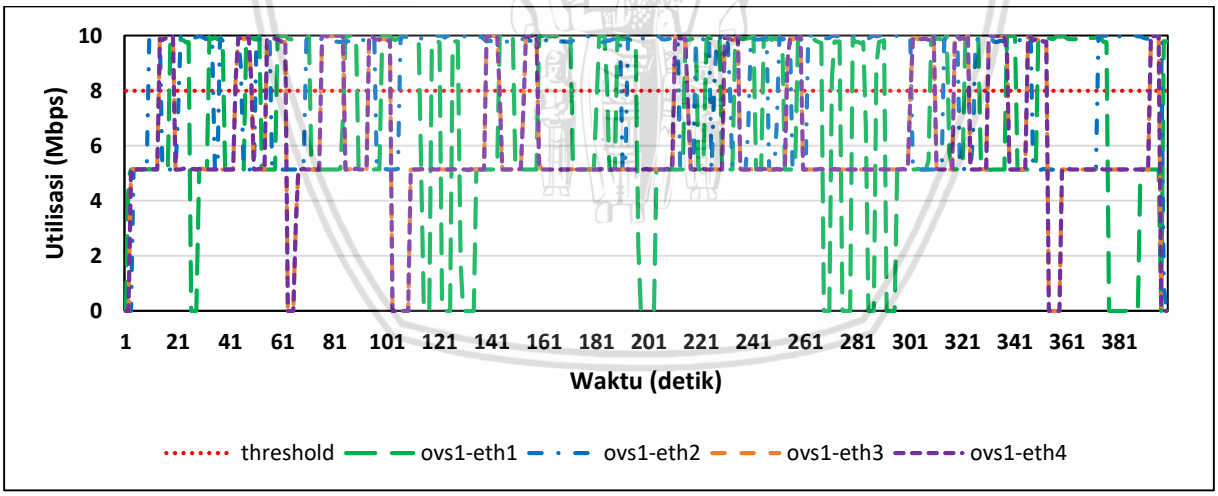
Gambar 5.1 adalah grafik pemerataan *traffic-flow* dari pengujian *throughput* pada metode Slavica. Terlihat pada grafik di atas terjadi proses perhitungan ulang berkali – kali. Proses perhitungan ulang dapat diketahui dari bentuk bagian grafik (gambar 5.2). Ketika terjadi perhitungan ulang maka segmen SlvSegTCP1 akan selalu mendapat *route-path* yang sama. Total kapasitas segmen SlvSegTCP1 adalah 10 Mbps, maka *route-path* akan mengalami *overload-path* dan menyebabkan perhitungan ulang. Hal tersebut terjadi secara terus menerus sehingga terjadi penurunan *throughput* sesaat secara terus – menerus. *Path-*

load pada ovs1-eth1 selalu melebihi batas *load-threshold* dan berada pada nilai rata – rata 10 Mbps atau maksimum *bandwidth path*. Sementara pada *path* ovs1-eth2, ovs1-eth3 dan ovs1-eth4 belum terutilisasi secara optimal.



Gambar 5.2. Grafik pada saat proses perhitungan ulang

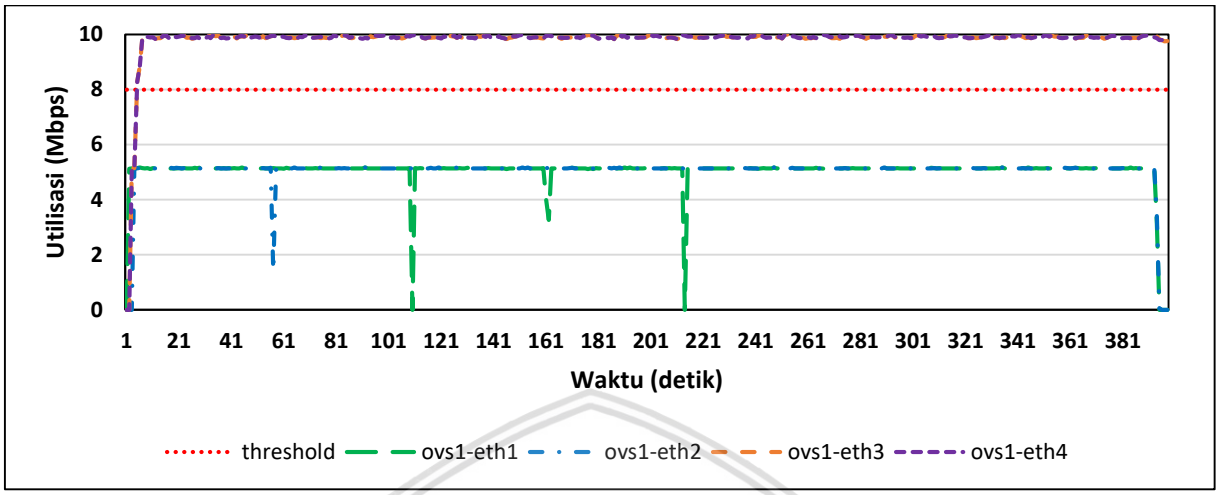
Pada gambar 5.2 bagian yang dilingkari dengan garis putus – putus merah merupakan keadaan *path-load* saat terjadi proses perhitungan ulang *route-path*. Pada saat terjadi proses perhitungan ulang, maka beberapa *table-flow* pada *switch* akan berubah dan itu berdampak pada utilisasi *path* akan mengalami penurunan dalam beberapa saat yang singkat.



Gambar 5.3. Grafik path load dengan metode LABERIO

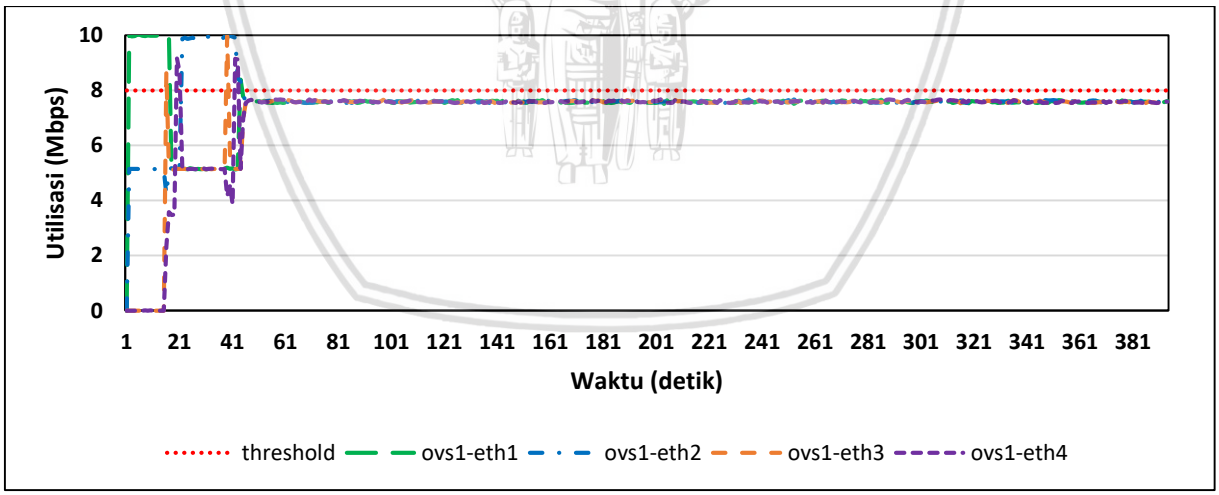
Gambar 5.3 adalah grafik pemerataan *traffic-flow* dari pengujian *throughput* pada metode LABERIO. Pada metode LABERIO terlihat sepanjang waktu terjadi perubahan *path-load*. Sama halnya pada metode Slavica, *overload-path* terjadi sepanjang percobaan. Hal ini disebabkan LbrSegTCP1 yang mendapat *route-path* yang sama. Jika menurut data segmentasi *traffic-flow*, maka setiap proses perhitungan ulang *route-path* pada *overload-path* maka akan terjadi satu sampai dengan dua *overload-path*. Metode LABERIO pada

pengujian *throughput* tidak dapat meratakan seluruh *traffic-flow* pada seluruh *path* yang tersedia.



Gambar 5.4. Grafik path-load dengan metode DLPO

Gambar 5.4 adalah grafik pemerataan *traffic-flow* dari pengujian *throughput* terhadap metode DLPO. Pada grafik 5.4 terlihat ovs1-eth3 dan ovs1-eth4 berada di atas batas *load-threshold*. Metode DLPO tidak merubah *route-path traffic-flow* pada saat terjadi *overload-path*, melainkan hanya merubah nilai prioritas pada *flow-table*. Ini menyebabkan *traffic-flow* tidak terbagi kepada *path* lain yang tersedia.



Gambar 5.5. Grafik path-load dengan metode Hybrid

Grafik pemerataan *traffic-flow* dari pengujian *throughput* pada metode Hybrid nampak pada gambar 5.5. Pada grafik di atas nampak metode Hybrid dapat melakukan pemerataan *traffic-flow* kepada seluruh *path* pada jaringan. Pada saat awal diberikan *traffic-flow*, metode Hybrid terjadi *overload-path* dan melakukan pembacaan data *monitoring*. Kemudian dilakukan proses perhitungan ulang *route-path* dan dilakukan penerapan *flow-*

table baru kepada seluruh *switch*. Proses perhitungan ulang terjadi dua kali dan kemudian seluruh *path* tidak mengalami *overload*. Pada saat perhitungan ulang *controller* akan bekerja secara *reactive* sehingga hanya sedikit terjadi penurunan *throughput*. Selain itu segmentasi yang lebih detail juga memberikan dampak positif bagi *throughput traffic-flow* secara keseluruhan.

5.2. Pengujian Latensi

Dengan mengacu pada konsep dan tujuan pengujian latensi pada bab 3 dan mengikuti langkah – langkah teknis pengujian pada bab 4 maka pada pengujian latensi diperoleh hasil pengujian yang ditampilkan pada tabel 5.3.

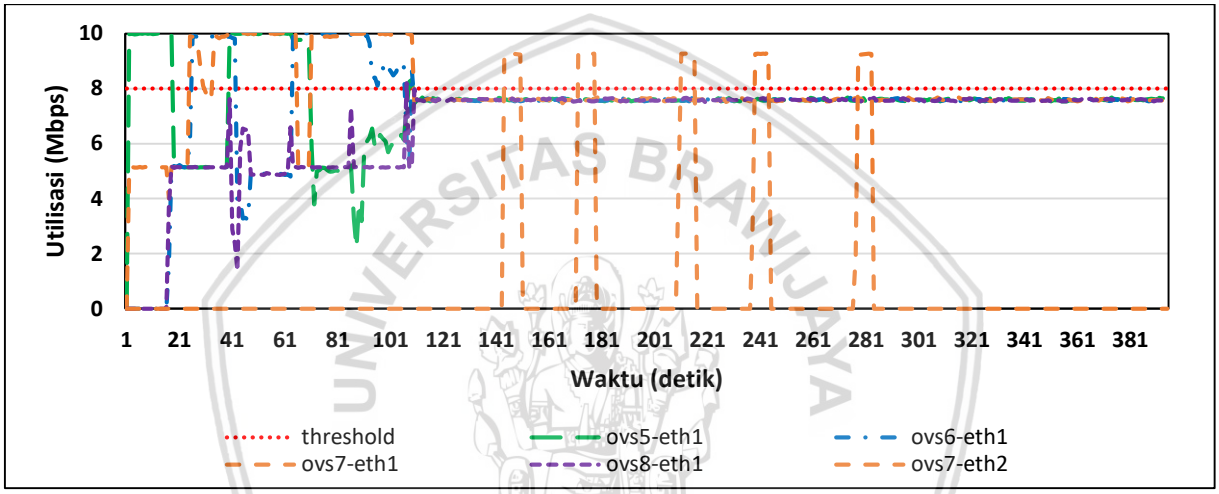
Tabel 5.3. Hasil pengujian latensi

Metode	Avg. Bw (Kbps)	Durasi (second)
Slavica	642.5	76
LABERIO	775.1	63
DLPO	739.8	66
Hybrid	1100	45

Tabel 5.3 adalah tabel hasil pengujian latensi pada seluruh metode. Metode Hybrid dapat menyelesaikan proses *copy* file dengan durasi lebih cepat dari pada metode yang lain. Perolehan *bandwidth* rata – rata pada saat proses *copy* file pada metode Hybrid adalah 1,1 Mbps. Perolehan *bandwidth* yang tinggi semakin mempercepat proses *copy* file. Pada metode LABERIO lebih lama 18 detik dan DLPO lebih lama 21 detik dari pada metode Hybrid. Sedangkan Slavica selisih 31 detik lebih lama dari pada metode Hybrid. Hal ini disebabkan karena pada metode Hybrid pemerataan *traffic-flow* (pengujian *throughput*) merata pada seluruh *path* sehingga ketika terdapat *traffic-flow* baru (pengujian latensi) akan memperoleh *throughput* lebih besar dibanding metode lainnya. Proses *copy* file menggunakan protokol TCP dengan *port number* 22. Dengan mode segmentasi yang digunakan pada metode Hybrid dapat memisahkan *traffic-flow* TCP ke beberapa *path* yang tersedia. Dengan demikian *latency* pada metode Hybrid lebih kecil jika dibanding dengan metode lainnya.

5.3. Pengujian Respon Sistem

Berdasarkan hasil pengujian *throughput*, metode Hybrid dapat meratakan beban *traffic-flow* kepada seluruh *path* pada jaringan sehingga beban *traffic-flow* pada masing – masing *path* tidak melebihi batas *load-threshold* (80% dari *bandwidth path*). Dengan demikian pegujian respon sistem terhadap *traffic*-spike hanya dilakukan pada metode Hybrid. Dengan mengacu pada konsep dan tujuan pengujian respon sistem pada bab 3 dan mengikuti langkah – langkah teknis pengujian pada bab 4 maka pada pengujian respon sistem diperoleh hasil pengujian yang ditampilkan pada gambar 5.6.



Gambar 5.6. Grafik pengujian respon sistem pada metode Hybrid

Gambar 5.6 adalah grafik utilisasi *path* untuk pengujian respon sistem terhadap metode Hybrid. Terlihat pada grafik di atas terjadi 5 kali *traffic-spike* yang sangat singkat. Pada saat *traffic-spike* terjadi, metode Hybrid tidak langsung merespon perubahan *traffic-flow*. Akan tetapi metode Hybrid melihat konsistensi *traffic-spike* tersebut. Apabila *traffic-spike* tersebut berakhir sebelum batas *time threshold* maka metode Hybrid tidak melakukan perhitungan ulang dan semua *path* kembali di bawah batas *load-threshold*.

BAB VI
KESIMPULAN DAN SARAN

6.1. Kesimpulan

Melihat pada hasil percobaan dan analisa yang dilakukan, maka dapat disimpulkan beberapa hal sebagai berikut :

1. Dengan menerapkan hibridasi mode proactive-reactive controller, metode Hybrid dapat secara proactive menentukan rute *traffic-flow* dan secara reactive dapat mendeteksi perubahan beban *traffic-flow* pada seluruh *path* dapat meratakan *traffic-flow* dibanding dengan metode Slavica, LABERIO dan DLPO.
2. Segmentasi *traffic-flow* yang detail dengan mempertimbangkan IP *address, internet protocol* dan *port number* membuat kapasitas tiap segmen semakin lebih kecil. Dengan kapasitas segmen yang kebih kecil akan memudahkan metode Hybrid dalam meratakan *traffic-flow* ke seluruh *path* pada jaringan.
3. Pembagian *traffic-flow* yang merata pada metode Hybrid memberikan dampak positif pada *throughput flow* sehingga metode Hybrid dapat meningkatkan *throughput flow* rata – rata sebesar 7,558 Mbps (25,193%) dari Slavica, 2,38 Mbps (7,93%) dari LABERIO dan 3,44 Mbps (11,47%) dari DLPO.
4. Dengan meningkatnya perolehan *throughput flow* memberikan dampak positif pada latensi pengiriman *flow*. Metode Hybrid menurunkan *latency* pengiriman *flow* rata – rata sebesar 31 detik dari Slavica, 18 detik dari LABERIO dan 21 detik dari DLPO.
5. Dengan melakukan *monitoring* kepada seluruh *path*, metode Hybrid dapat mendeteksi terjadinya perubahan kapasitas *traffic-flow* dan dengan menerapkan *time-threshold* selama 15 detik dapat mendeteksi terjadinya *traffic-spike*.

6.2. Saran

Dari pengalaman percobaan pada penelitian ini, berikut beberapa saran untuk penelitian selanjutnya :

1. Pemisahan *resource* antara *controller* dengan aplikasi deteksi dan perhitungan. Hal ini dianggap perlu untuk meningkatkan kecepatan proses perhitungan dan performa *controller*.
2. Pada penelitian selanjutnya dapat dilakukan optimasi sistem *monitoring* dan sistem prediksi perilaku trafik pada jaringan berdasar data periodik *monitoring*.
3. Pembuatan *environment* yang terbuka dan bebas untuk diakses yang berisikan metode – metode dari penelitian sebelumnya guna untuk mempercepat proses penelitian selanjutnya.

DAFTAR PUSTAKA

Al-Fares, M. (2008). *A scalable commodity data center network architecture.* New York: ACM SIGCOMM 2008 conference on Data communication.

Architecture framework for the development of signalling and OA&M protocols using OSI concepts. (1993). Retrieved from International Telecommunication Union: http://www.itu.int/rec/T-REC-Q.1400/en/

Bilal, K. (2014). A comparative analysis of data center network architectures. *Proceedings 26th European Conference on Modelling and Simulation ©ECMS* (p. 10.1109/ICC.2014.6883798). Sydney: IEEE.

Fitria, A. T. (2013). Implementasi Algoritma Dijkstra Dalam Aplikasi Untuk Menentukan Lintasan Terpendek Jalan Darat Antar Kota Di Sumatera Bagian Selatan. *Jurnal Sistem Informasi (JSI)*.

Fortz, B. (2002, October). Traffic Engineering with Traditional IP Routing Protocols. *IEEE Communications Magazine*, p. 7422923.

Gropp, V. (2015, August 2). *bwm-ng (Bandwidth Monitor NG).* Retrieved from homepage of volker gropp: https://www.gropp.org/?id=projects&sub=bwm-ng

Hou, W. (2017). An improved SDN-based fabric for flexible data center networks. *Computing, Networking and Communications (ICNC), 2017 International Conference on* (p. 7876167). Santa Clara: IEEE.

http://mininet.org/. (2016). *Documentation*. Retrieved from An Instant Virtual Network on your Laptop (or other PC): http://mininet.org/

Hui Long, Y. S. (2013). LABERIO: Dynamic load-balanced Routing in OpenFlow-enabled Networks. *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)* (p. 13581045). Barcelona: IEEE.

IANA. (2017, 11 16). *Service Name and Transport Protocol Port Number Registry.* Retrieved from Internet Assigned Numbers Authority: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

Internet Engineering Task Force. (2000, November). *Analysis of an Equal-Cost Multi-Path Algorithm.* Retrieved from Internet Engineering Task Force: https://tools.ietf.org/rfc/rfc2992.txt

Internet Engineering Task Force. (2003, July). *RTP: A Transport Protocol for Real-Time Applications.* Retrieved from Internet Engineering Task Force: https://www.ietf.org/rfc/rfc3550.txt

Internet Engineering Task Force. (2005, May). *Fast Reroute Extensions to RSVP-TE for LSP Tunnels.* Retrieved from Internet Engineering Task Force: https://tools.ietf.org/rfc/rfc4090.txt

Jehn-Ruey Jiang, H.-W. H.-H.-Y. (2014). Extending Dijkstra's Shortest Path Algorithm for Software Defined Networking. *IEEE*.

Kayri, M. (2010). A PROPOSED "OSI BASED" NETWORK TROUBLES IDENTIFICATION MODEL . *International Journal of Next-Generation Networks (IJNGN)*, 3.

Komunitas SDN-RG - Institut Teknologi Bandung. (2014). Pengantar SDN. In M. A. Aris Cahyadi Risdianto, *Buku Komunitas SDN-RG*. Bandung: Institut Teknologi Bandung.

Lazuardi, E. (2016). *Metode Pemilihan Jalur Routing Adaptif Berdasar Kemacetan Jaringan Dengan Algoritma Dijkstra Pada Openflow Network.* Malang: Program Studi Teknik Informatika Universitas Brawijaya.

Mikrotik Indonesia. (n.d.). *Konfigurasi Dasar OSPF*. Retrieved from Mikrotik Indonesia: http://mikrotik.co.id/

MultiPath TCP - Linux Kernel implementation. (2012). *Welcome to the Linux Kernel MultiPath TCP project*. Retrieved from MultiPath TCP - Linux Kernel implementation: http://multipath-tcp.org/

NetworkX. (2017, September 20). *NetworkX*. Retrieved from NetworkX: https://networkx.github.io/

olegslavkin. (2016). *Install Ryu Controller (Ubuntu 14.04.3 Server)*. Retrieved from Ryu Controller: https://gist.github.com/olegslavkin/e01ccc1835396402dc2f

Open Network Foundation. (2012). *Open Network Foundation*. Retrieved from Open Network Foundation: https://www.opennetworking.org

Open Networking Foundation. (2012). *ONF White Paper*. Retrieved from Open Networking Foundation: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

R. Mohtasin, P. P. (2016). Development of a Virtualized Networking Lab using GNS3 and VMware Workstation. *Wireless Communications, Signal Processing and Networking (WiSPNET), International Conference on.* IEEE.

RYU. (2014). *RYU SDN Framework*. Retrieved from RYU SDN Framework: https://osrg.github.io/ryu-book/en/html/

SDX Central. (2015). *Ryu Controller*. Retrieved from SDX Central: https://www.sdxcentral.com

Shafqat Ur Rehman, W.-C. S. (2014). Network-Wide Traffic Visibility in OF@TEIN SDN Testbed using sFlow. *Network Operations and Management Symposium (APNOMS), Asia-Pacific.* IEEE.

Shie-Yuan, H.-Y. P. (2015). Optimizing the SDN Control-Plane Performance of the Openvswitch Software Switch. *IEEE Symposium on Computers and Communication (ISCC).*

Shih-Chun Lin, I. F. (2016). QoS-aware Adaptive Routing in Multi-layer Hierarchical Software Defined Networks: A Reinforcement Learning Approach. *2016 IEEE International Conference on Services Computing.* IEEE.

Slavica Tomovic, N. L. (2016). A new approach to dynamic routing in SDN networks. *Proceedings of the 18th Mediterranean Electrotechnical Conference.* MELECON: IEEE.

Sudha. (2013). Performance Analysis of Kernel-Based Virtual Machine. *International Journal of Computer Science & Information Technology (IJCSIT) Vol 5,*, No 1.

Suherman. (2011). *Simulasi Algoritma Dijkstra Pada Protokol Routing Open Shortest Path First.* Diponegoro University.

The ICSI Networking and Security Group. (1999). *The TCP-Friendly Website*. Retrieved from The ICSI Networking and Security Group: http://www.icir.org/floyd/tcp_friendly.html

The TCP/IP Guide. (2005). *TCP Adaptive Retransmission and Retransmission Timer Calculations*. Retrieved from The TCP/IP Guide: http://www.tcpipguide.com/free/t_TCPAdaptiveRetransmissionandRetransmissionTimerCal.htm

WSGI.org. (2016, December 6). *WSGI*. Retrieved from WSGI: http://wsgi.readthedocs.io/en/latest/

Yi-Chih Lei, K. W.-H. (2015). Multipath Routing in SDN-based Data Center Networks . *IEEE*.

Yuan-Liang Lan, K. W.-H. (2016). Dynamic Load-balanced Path Optimization in SDN-based Data Center Networks. *2016 10th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)* (p. 7573945). Prague: IEEE.

LAMPIRAN

Lampiran 1. Script python fat-tree-A.py

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():
      net = Mininet( topo=None,
                           build=False,
                           ipBase='10.0.0.0/8')

      info( '*** Adding controller\n' )
      c0=net.addController(name='c0',
                                 controller=RemoteController,
                                 ip='127.0.0.1',
                                 protocol='tcp',
                                 port=6633)

      info( '*** Add switches\n')
      ovs1 = net.addSwitch('ovs1', cls=OVSKernelSwitch, dpid='1')
      ovs2 = net.addSwitch('ovs2', cls=OVSKernelSwitch, dpid='2')
      ovs3 = net.addSwitch('ovs3', cls=OVSKernelSwitch, dpid='3')
      ovs4 = net.addSwitch('ovs4', cls=OVSKernelSwitch, dpid='4')
      ovs5 = net.addSwitch('ovs5', cls=OVSKernelSwitch, dpid='5')
      ovs6 = net.addSwitch('ovs6', cls=OVSKernelSwitch, dpid='6')
      ovs7 = net.addSwitch('ovs7', cls=OVSKernelSwitch, dpid='7')
      ovs8 = net.addSwitch('ovs8', cls=OVSKernelSwitch, dpid='8')
      ovs9 = net.addSwitch('ovs9', cls=OVSKernelSwitch, dpid='9')
      ovs10 = net.addSwitch('ovs10', cls=OVSKernelSwitch, dpid='a')

      info( '*** Add hosts\n')
      h1 = net.addHost('h1', cls=Host, ip='10.0.2.1/8',
defaultRoute=None)
      h2 = net.addHost('h2', cls=Host, ip='10.0.2.2/8',
defaultRoute=None)
      h3 = net.addHost('h3', cls=Host, ip='10.0.2.3/8',
defaultRoute=None)
      h4 = net.addHost('h4', cls=Host, ip='10.0.2.4/8',
defaultRoute=None)
      h5 = net.addHost('h5', cls=Host, ip='10.0.2.5/8',
defaultRoute=None)
      h6 = net.addHost('h6', cls=Host, ip='10.0.2.6/8',
defaultRoute=None)
      h7 = net.addHost('h7', cls=Host, ip='10.0.2.7/8',
defaultRoute=None)
```

```
      h8 = net.addHost('h8', cls=Host, ip='10.0.2.8/8',
defaultRoute=None)
      h9 = net.addHost('h9', cls=Host, ip='10.0.2.9/8',
defaultRoute=None)
      h10 = net.addHost('h10', cls=Host, ip='10.0.2.10/8',
defaultRoute=None)
      h11 = net.addHost('h11', cls=Host, ip='10.0.2.11/8',
defaultRoute=None)
      h12 = net.addHost('h12', cls=Host, ip='10.0.2.12/8',
defaultRoute=None)
      h13 = net.addHost('h13', cls=Host, ip='10.0.2.13/8',
defaultRoute=None)
      h14 = net.addHost('h14', cls=Host, ip='10.0.2.14/8',
defaultRoute=None)
      h15 = net.addHost('h15', cls=Host, ip='10.0.2.15/8',
defaultRoute=None)
      h16 = net.addHost('h16', cls=Host, ip='10.0.2.16/8',
defaultRoute=None)
      h17 = net.addHost('h17', cls=Host, ip='10.0.2.17/8',
defaultRoute=None)
      h18 = net.addHost('h18', cls=Host, ip='10.0.2.18/8',
defaultRoute=None)
      h19 = net.addHost('h19', cls=Host, ip='10.0.2.19/8',
defaultRoute=None)
      h20 = net.addHost('h20', cls=Host, ip='10.0.2.20/8',
defaultRoute=None)

      info( '*** Add links\n')
      net.addLink(ovs1, ovs5, cls=TCLink, bw=10)
      net.addLink(ovs1, ovs6, cls=TCLink, bw=10)
      net.addLink(ovs1, ovs7, cls=TCLink, bw=10)
      net.addLink(ovs1, ovs8, cls=TCLink, bw=10)

      net.addLink(ovs2, ovs5, cls=TCLink, bw=10)
      net.addLink(ovs2, ovs6, cls=TCLink, bw=10)
      net.addLink(ovs2, ovs7, cls=TCLink, bw=10)
      net.addLink(ovs2, ovs8, cls=TCLink, bw=10)

      net.addLink(ovs3, ovs5, cls=TCLink, bw=10)
      net.addLink(ovs3, ovs6, cls=TCLink, bw=10)
      net.addLink(ovs3, ovs7, cls=TCLink, bw=10)
      net.addLink(ovs3, ovs8, cls=TCLink, bw=10)

      net.addLink(ovs4, ovs5, cls=TCLink, bw=10)
      net.addLink(ovs4, ovs6, cls=TCLink, bw=10)
      net.addLink(ovs4, ovs7, cls=TCLink, bw=10)
      net.addLink(ovs4, ovs8, cls=TCLink, bw=10)

      net.addLink(ovs5, ovs9, cls=TCLink, bw=10)
      net.addLink(ovs5, ovs10, cls=TCLink, bw=10)
      net.addLink(ovs6, ovs9, cls=TCLink, bw=10)
      net.addLink(ovs6, ovs10, cls=TCLink, bw=10)
      net.addLink(ovs7, ovs9, cls=TCLink, bw=10)
      net.addLink(ovs7, ovs10, cls=TCLink, bw=10)
      net.addLink(ovs8, ovs9, cls=TCLink, bw=10)
      net.addLink(ovs8, ovs10, cls=TCLink, bw=10)

      net.addLink(ovs1, h1, cls=TCLink, bw=10)
      net.addLink(ovs1, h2, cls=TCLink, bw=10)
```

```
net.addLink(ovs1, h3, cls=TCLink, bw=10)
net.addLink(ovs1, h4, cls=TCLink, bw=10)
net.addLink(ovs1, h5, cls=TCLink, bw=10)

net.addLink(ovs2, h6, cls=TCLink, bw=10)
net.addLink(ovs2, h7, cls=TCLink, bw=10)
net.addLink(ovs2, h8, cls=TCLink, bw=10)
net.addLink(ovs2, h9, cls=TCLink, bw=10)
net.addLink(ovs2, h10, cls=TCLink, bw=10)

net.addLink(ovs3, h11, cls=TCLink, bw=10)
net.addLink(ovs3, h12, cls=TCLink, bw=10)
net.addLink(ovs3, h13, cls=TCLink, bw=10)
net.addLink(ovs3, h14, cls=TCLink, bw=10)
net.addLink(ovs3, h15, cls=TCLink, bw=10)

net.addLink(ovs4, h16, cls=TCLink, bw=10)
net.addLink(ovs4, h17, cls=TCLink, bw=10)
net.addLink(ovs4, h18, cls=TCLink, bw=10)
net.addLink(ovs4, h19, cls=TCLink, bw=10)
net.addLink(ovs4, h20, cls=TCLink, bw=10)

info( '*** Starting network\n')
net.build()
info( '*** Starting controllers\n')
for controller in net.controllers:
      controller.start()

info( '*** Starting switches\n')
net.get('ovs1').start([c0])
net.get('ovs2').start([c0])
net.get('ovs3').start([c0])
net.get('ovs4').start([c0])
net.get('ovs5').start([c0])
net.get('ovs6').start([c0])
net.get('ovs7').start([c0])
net.get('ovs8').start([c0])
net.get('ovs9').start([c0])
net.get('ovs10').start([c0])

info( '*** Post configure switches and hosts\n')
ovs1.cmd('ifconfig ovs1 10.0.1.1/8')
ovs2.cmd('ifconfig ovs2 10.0.1.2/8')
ovs3.cmd('ifconfig ovs3 10.0.1.3/8')
ovs4.cmd('ifconfig ovs4 10.0.1.4/8')
ovs5.cmd('ifconfig ovs5 10.0.1.5/8')
ovs6.cmd('ifconfig ovs6 10.0.1.6/8')
ovs7.cmd('ifconfig ovs7 10.0.1.7/8')
ovs8.cmd('ifconfig ovs8 10.0.1.8/8')
ovs9.cmd('ifconfig ovs9 10.0.1.9/8')
ovs10.cmd('ifconfig ovs10 10.0.1.10/8')

h1.cmdPrint('ping 10.0.0.1 -c 1 -W 1')
h2.cmdPrint('ping 10.0.0.1 -c 1 -W 1')
h3.cmdPrint('ping 10.0.0.1 -c 1 -W 1')
h4.cmdPrint('ping 10.0.0.1 -c 1 -W 1')
h5.cmdPrint('ping 10.0.0.1 -c 1 -W 1')
h6.cmdPrint('ping 10.0.0.1 -c 1 -W 1')
h7.cmdPrint('ping 10.0.0.1 -c 1 -W 1')
```

```
        h8.cmdPrint('ping 10.0.0.1 -c 1 -W 1')
        h9.cmdPrint('ping 10.0.0.1 -c 1 -W 1')
        h10.cmdPrint('ping 10.0.0.1 -c 1 -W 1')
        h11.cmdPrint('ping 10.0.0.1 -c 1 -W 1')
        h12.cmdPrint('ping 10.0.0.1 -c 1 -W 1')
        h13.cmdPrint('ping 10.0.0.1 -c 1 -W 1')
        h14.cmdPrint('ping 10.0.0.1 -c 1 -W 1')
        h15.cmdPrint('ping 10.0.0.1 -c 1 -W 1')
        h16.cmdPrint('ping 10.0.0.1 -c 1 -W 1')
        h17.cmdPrint('ping 10.0.0.1 -c 1 -W 1')
        h18.cmdPrint('ping 10.0.0.1 -c 1 -W 1')
        h19.cmdPrint('ping 10.0.0.1 -c 1 -W 1')
        h20.cmdPrint('ping 10.0.0.1 -c 1 -W 1')

        call('ovs-vsctl -- --id=@MiniEditSF create sFlow
target=\"127.0.0.1\:6343\" header=128 sampling=400 polling=20 -- set
Bridge ovs1 sflow=@MiniEditSF -- set Bridge ovs4 sflow=@MiniEditSF -- set
Bridge ovs10 sflow=@MiniEditSF -- set Bridge ovs3 sflow=@MiniEditSF --
set Bridge ovs2 sflow=@MiniEditSF -- set Bridge ovs6 sflow=@MiniEditSF --
set Bridge ovs9 sflow=@MiniEditSF -- set Bridge ovs5 sflow=@MiniEditSF --
set Bridge ovs8 sflow=@MiniEditSF -- set Bridge ovs7 sflow=@MiniEditSF',
shell=True)

        CLI(net)
        net.stop()

if __name__ == '__main__':
        setLogLevel( 'info' )
        myNetwork()
```

Lampiran 2. Script Ryu controller dsi-ryu.py

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER,
DEAD_DISPATCHER
from ryu.ofproto.ofproto_v1_3 import OFPG_ANY
from ryu.ofproto.ofproto_v1_3 import OFP_VERSION
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types
from ryu.lib.packet import arp
from ryu.lib.packet import ipv4
from ryu.lib.packet import tcp
from ryu.lib.packet import udp
from ryu.app.wsgi import ControllerBase, WSGIApplication, route
from ryu.lib import dpid as dpid_lib
from ryu.topology import event, switches
from ryu.topology.api import get_switch, get_link, get_host

from webob import Response
import array
import json
import signal
import socket
import urllib
import asyncore
import ConfigParser
import argparse
import os
import sys
import datetime
from datetime import timedelta
import logging
from logging.handlers import TimedRotatingFileHandler
import math
import time
import copy
from classDB import MyDatabase
from classIPConverter import *
import networkx as nx
from networkx.readwrite import json_graph
from networkx import *
import threading
import matplotlib
import matplotlib.pyplot as plt

MyDynRouteAppInstanceName = 'controller_ryu'
MyDynRouteAppBaseUrl = '/controller/'
class MySwitchMode:
      Controller=1
      StandAlone=2

class MyDynRouteApp(app_manager.RyuApp):
      OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
      _CONTEXTS = {'wsgi': WSGIApplication}
```

```
	def __init__(self, *args, **kwargs):
		super(MyDynRouteApp, self).__init__(*args, **kwargs)
		wsgi = kwargs['wsgi']

	wsgi.register(MyDynRouteController,{MyDynRouteAppInstanceName:
self})

		self.mSwitch={}
		self.mGraph = None
		self.mMapIpMac = {}
		self.mRouteHistory = {}

		currentDir = os.path.dirname(os.path.realpath('__file__'))
		configFile = currentDir + '/dsi-config.conf'
		self.mConfig = ConfigParser.ConfigParser()
		self.mConfig.read(configFile)

		self.mAppUrl = self.mConfig.get('general','app_url')
		self.mUseRouteHistory =
self.mConfig.getboolean('general','use_route_history')

		dbHost = self.mConfig.get('database','host')
		dbUser = self.mConfig.get('database','user')
		dbPassword = self.mConfig.get('database','password')
		dbName = self.mConfig.get('database','name')
		logFile = self.mConfig.get('logging','controller_log_file')
		logLevel = self.mConfig.get('logging','controller_log_level')
		self.mLogger = logging.getLogger(__name__)
		logHdl = logging.handlers.TimedRotatingFileHandler(logFile,
when='midnight', backupCount=10)
		logFormatter = logging.Formatter('%(asctime)s %(levelname)s
%(message)s')
		logHdl.setFormatter(logFormatter)
		self.mLogger.addHandler(logHdl)
		if(logLevel == 'CRITICAL'):
			self.mLogger.setLevel(logging.CRITICAL)
		elif(logLevel == 'ERROR'):
			self.mLogger.setLevel(logging.ERROR)
		elif(logLevel == 'WARNING'):
			self.mLogger.setLevel(logging.WARNING)
		elif(logLevel == 'INFO'):
			self.mLogger.setLevel(logging.INFO)
		elif(logLevel == 'DEBUG'):
			self.mLogger.setLevel(logging.DEBUG)
		else:
			self.mLogger.setLevel(logging.DEBUG)

		self.mDatabase = MyDatabase(host=dbHost, user=dbUser,
password=dbPassword, database=dbName, logObj=self.mLogger)
		self.mDatabase.connect()
		self.mDatabase.disconnect()

	@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
	def switch_features_handler(self, ev):
		self.mLogger.info('switch_features_handler '+str(ev))
		datapath = ev.msg.datapath
		self.mLogger.info('detected switch dpid '+str(datapath.id))
		self.mSwitch[datapath.id] = datapath
```

```
            self.emptyFlowTable(datapath,0)
            self.addDefaultFlow(datapath, MySwitchMode.Controller)

      @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
      def _packet_in_handler(self, ev):
            #print '_packet_in_handler '+str(ev)
            #self.mLogger.info('_packet_in_handler '+str(ev))
            msg = ev.msg
            datapath = msg.datapath
            ofproto = datapath.ofproto
            parser = datapath.ofproto_parser
            in_port = msg.match['in_port']

            pkt = packet.Packet(msg.data)
            eth = pkt.get_protocols(ethernet.ethernet)[0]
            if eth.ethertype == ether_types.ETH_TYPE_LLDP:
                  return

            #print pkt

            srcHwAddr = eth.src
            dstHwAddr = eth.dst
            srcIpAddr = None
            dstIpAddr = None
            IpProto = None
            if(pkt.get_protocols(arp.arp)):
                  srcIpAddr = pkt.get_protocols(arp.arp)[0].src_ip
                  dstIpAddr = pkt.get_protocols(arp.arp)[0].dst_ip

            if(srcIpAddr == None):
                  if(pkt.get_protocols(ipv4.ipv4)):
                        srcIpAddr = pkt.get_protocols(ipv4.ipv4)[0].src
                        dstIpAddr = pkt.get_protocols(ipv4.ipv4)[0].dst
                        IpProto = pkt.get_protocols(ipv4.ipv4)[0].proto

            if(dstIpAddr == '224.0.0.251'):
                  return

            print 'CONTROLLER REACTIVE HANDLING'

            if(self.mGraph is not None):
                  #print srcHwAddr+'('+srcIpAddr+') =>
'+dstHwAddr+'('+dstIpAddr+')'
                  srcNodeId = srcHwAddr
                  if(dstHwAddr == 'ff:ff:ff:ff:ff:ff'):
                        dstNodeId = self.mMapIpMac[dstIpAddr]
                  else:
                        dstNodeId = dstHwAddr

                  pRouteNode = None
                  try:
                        if self.mUseRouteHistory:
                              pRouteHistoryId = srcNodeId+'-'+dstNodeId
                              if (pRouteHistoryId in self.mRouteHistory):
                                    pRouteNode =
self.mRouteHistory[pRouteHistoryId]
                              else:
                                    pRouteNode =
nx.dijkstra_path(self.mGraph,srcNodeId,dstNodeId)
```

```
                                    self.mRouteHistory[pRouteHistoryId] =
pRouteNode
                        else:
                              pRouteNode =
nx.dijkstra_path(self.mGraph,srcNodeId,dstNodeId)
                  except:
                        #self.mLogger.info('cannot route '+srcNodeId+' to
'+dstNodeId)
                        pass

                  if pRouteNode is not None:
                        for i in range(0, len(pRouteNode)):
                              if pRouteNode[i] == str(datapath.id):
                                    srcEdgeNodeId = str(datapath.id)
                                    dstEdgeNodeId = pRouteNode[i+1]
                                    dtNextEdge =
self.mGraph.get_edge_data(srcEdgeNodeId, dstEdgeNodeId)
                                    #print dtNextEdge
                                    outPort =
dtNextEdge['data']['src_intf_port_no']
                                    #print outPort
                                    actions =
[parser.OFPActionOutput(outPort)]
                                    data = None
                                    if msg.buffer_id ==
ofproto.OFP_NO_BUFFER:
                                          data = msg.data
                                    out =
parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,

      in_port=in_port, actions=actions, data=data)
                                    datapath.send_msg(out)

            else:
                  #print 'no graph loaded'
                  pass

      def discoveryNetwork(self):
            strSql = 'delete from `t_switch`'
            self.mDatabase.execQuery(strSql)
            strSql = 'delete from `t_host`'
            self.mDatabase.execQuery(strSql)
            strSql = 'delete from `t_intf`'
            self.mDatabase.execQuery(strSql)
            strSql = 'delete from `t_link`'
            self.mDatabase.execQuery(strSql)

            self.mMapIpMac = {}
            switch_list = get_switch(self, None)
            for switch in switch_list:
                  strSql = 'insert into t_switch (`id`,`dpid`) values
("'+str(switch.dp.id)+'",'+str(switch.dp.id)+')'
                  self.mDatabase.execQuery(strSql)
                  for port in switch.ports:
                        #print port
                        strSql = 'insert into t_intf
(`id`,`hw_addr`,`name`,`dpid`,`port_no`) values ('+ \
                              '"'+port.hw_addr+'",'+ \
                              '"'+port.hw_addr+'",'+ \
```

```
                                    '"'+port.name+'",'+ \
                                    str(port.dpid)+','+ \
                                    str(port.port_no)+ \
                                    ')'
                              self.mDatabase.execQuery(strSql)

                        hosts = get_host(self,switch.dp.id)
                        #print str(switch.dp.id)+'>>>>>>>>>>>>>>>'
                        for host in hosts:
                              self.mMapIpMac[host.ipv4[0]] = host.mac
                              shortIpLabel =
host.ipv4[0].split('.')[2]+'.'+host.ipv4[0].split('.')[3]

                              #print vars(host)
                              strSql = 'insert into t_host
(`id`,`hw_addr`,`ipv4`,`label`) values
("'+host.mac+'","'+host.mac+'","'+host.ipv4[0]+'","'+shortIpLabel+'")'
                              self.mDatabase.execQuery(strSql)
                              strSql = 'insert into t_intf (`id`,`hw_addr`)
values ("'+host.mac+'","'+host.mac+'")'
                              self.mDatabase.execQuery(strSql)

                              #print vars(host.port)
                              strSql = 'insert into t_link
(`src_intf_id`,`dst_intf_id`,`bw`) values ('+ \
                                    '"'+host.port.hw_addr+'",'+ \
                                    '"'+host.mac+'",'+ \
                                    '0'+ \
                                    ')'
                              self.mDatabase.execQuery(strSql)
                              strSql = 'insert into t_link
(`src_intf_id`,`dst_intf_id`,`bw`) values ('+ \
                                    '"'+host.mac+'",'+ \
                                    '"'+host.port.hw_addr+'",'+ \
                                    '0'+ \
                                    ')'
                              self.mDatabase.execQuery(strSql)

                  links_list = get_link(self, None)
                  for link in links_list:
                        #print '======================'
                        #print vars(link.src)
                        #print '======================'
                        strSql = 'insert into t_link
(`src_intf_id`,`dst_intf_id`,`bw`) values ('+ \
                              '"'+link.src.hw_addr+'",'+ \
                              '"'+link.dst.hw_addr+'",'+ \
                              '0'+ \
                              ')'
                        self.mDatabase.execQuery(strSql)

                  self.mGraph = self.loadGraph()
                  return self.getGraphData()

            def loadGraph(self):
                  self.mLogger.info('load data graph')
                  pGraph = nx.DiGraph()
                  ar1 = self.mDatabase.getArrayResult('select * from `v_node`')
                  for pRecord in ar1:
```

```
                if not pRecord['id'] in pGraph.node:
                        pGraph.add_node(pRecord['id'])
                        pGraph.node[pRecord['id']]['data'] = pRecord
                        pGraph.node[pRecord['id']]['intf'] = {}

        ar1 = self.mDatabase.getArrayResult('select * from `v_intf`')
        for pRecord in ar1:
                if pRecord['node_id'] in pGraph.node:

    pGraph.node[pRecord['node_id']]['intf'][pRecord['id']] = pRecord

        ar1 = self.mDatabase.getArrayResult('select * from `v_link`')
        for pRecord in ar1:
                #pWeight = self.mMaxLinkBw / pRecord['bw']
                pWeight = 0
                #print 'Add edge
'+pRecord['device_id_src']+'==>'+pRecord['device_id_dst']

    pGraph.add_edge(pRecord['src_node_id'],pRecord['dst_node_id'],
weight=pWeight, \
                                        load=0, data=pRecord)

        #self.saveGraphToImageFile(pGraph)

        #for node in pGraph.node:
        #    print pGraph.node[node]
        return pGraph

    def getGraphData(self):
        return json_graph.node_link_data(self.mGraph)

    def saveGraphToImageFile(self, pGraph):
        pos=nx.spectral_layout(pGraph)
        labels={}
        for xnode in pGraph.nodes():
                labels[xnode]=pGraph.node[xnode]['data']['label']
                #labels[xnode]=xnode

        #nx.draw(pGraph)
        nx.draw_networkx_nodes(pGraph,pos)
        nx.draw_networkx_edges(pGraph,pos,width=1.0,alpha=0.5)
        nx.draw_networkx_labels(pGraph,pos,labels)
        plt.axis('off')
        #plt.savefig("simple_path.png")
        plt.show()
        pass

    def addFlow(self, datapath, table_id, priority, match, actions):
        try:
                ofproto = datapath.ofproto
                parser = datapath.ofproto_parser
                inst =
[parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,

actions)]
                mod = parser.OFPFlowMod(datapath=datapath,
table_id=table_id, priority=priority,
                                        match=match,
instructions=inst)
```

```
                if datapath.send_msg(mod):
                        self.mLogger.info('add flow to
datapath='+str(datapath.id)+'; match='+str(match)+';
action='+str(actions)+'; OK')
                else:
                        self.mLogger.info('add flow to
datapath='+str(datapath.id)+'; match='+str(match)+';
action='+str(actions)+'; FAIL')
            except Exception, pError:
                self.mLogger.error(str(pError))
                self.mLogger.error('add flow to
datapath='+str(datapath.id)+'; match='+str(match)+';
action='+str(actions)+'; ERROR')

      def addDefaultFlow(self, datapath, switchMode):
            #self.emptyFlowTable(self.mSwitch[datapath.id],2)
            ofproto = datapath.ofproto
            parser = datapath.ofproto_parser
            match = parser.OFPMatch()
            actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,

ofproto.OFPCML_NO_BUFFER)]
            self.addFlow(datapath, 0, 2, match, actions)

      def emptyFlowTable(self, datapath, table_id):
            try:
                parser = datapath.ofproto_parser
                ofproto = datapath.ofproto
                empty_match = parser.OFPMatch()
                instructions = []
                flow_mod = datapath.ofproto_parser.OFPFlowMod(
      datapath = datapath,

                table_id = table_id,

                command = ofproto.OFPFC_DELETE,

                out_port = ofproto.OFPP_ANY,

                out_group = ofproto.OFPG_ANY,

                match = empty_match,

                instructions = instructions)
                if datapath.send_msg(flow_mod):
                        self.mLogger.info('empty flow tables on
datapath='+str(datapath.id)+'; OK')
                        if(table_id == 0):
                                self.addDefaultFlow(datapath,
MySwitchMode.Controller)
                else:
                        self.mLogger.info('empty flow tables on
datapath='+str(datapath.id)+'; FAIL')
            except Exception, pError:
                self.mLogger.error(str(pError))
                self.mLogger.error('empty flow tables on
datapath='+str(datapath.id)+'; ERROR')

      def applyFlowTableFromDb(self, pSwitchId):
```

```
            self.mLogger.info('start apply flow table from database...')
            self.mLogger.info('load switch '+pSwitchId+'...')
            arSwitchId = []
            if pSwitchId == 'ALL':
                  for i in self.mSwitch:
                        arSwitchId.append(i)
            else:
                  if int(pSwitchId) in self.mSwitch:
                        arSwitchId.append(int(pSwitchId))

            for switchId in arSwitchId:
                  self.emptyFlowTable(self.mSwitch[switchId], 0)
                  ofproto = self.mSwitch[switchId].ofproto
                  parser = self.mSwitch[switchId].ofproto_parser
                  self.mLogger.info('applying flow table from database
for switch '+str(switchId)+'...')
                  strSql = 'select * from `t_flow_table` where '+ \
                                    '`switch_id`="'+str(switchId)+'"'
                  dtFlowTable1 = self.mDatabase.getArrayResult(strSql)
                  for datax in dtFlowTable1:

                        dtFlowMatch = {}
                        dtFlowMatch['eth_type'] = 0x0800
                        dtFlowMatch['ipv4_src'] = datax['src_ipv4']
                        dtFlowMatch['ipv4_dst'] = datax['dst_ipv4']
                        if(datax['proto'] != -1):
                              dtFlowMatch['ip_proto'] = datax['proto']
                        if(datax['src_proto_port'] != -1):
                              if(datax['proto'] == 6):
                                    dtFlowMatch['tcp_src'] =
datax['src_proto_port']
                              elif(datax['proto'] == 17):
                                    dtFlowMatch['udp_src'] =
datax['src_proto_port']
                              elif(datax['proto'] == 132):
                                    dtFlowMatch['sctp_src'] =
datax['src_proto_port']
                        if(datax['dst_proto_port'] != -1):
                              if(datax['proto'] == 6):
                                    dtFlowMatch['tcp_dst'] =
datax['dst_proto_port']
                              elif(datax['proto'] == 17):
                                    dtFlowMatch['udp_dst'] =
datax['dst_proto_port']
                              elif(datax['proto'] == 132):
                                    dtFlowMatch['sctp_dst'] =
datax['dst_proto_port']

                        #memilih priority
                        wildcardProtoAndPort = False
                        if((datax['proto'] != -1) and
((datax['src_proto_port'] != -1) or (datax['dst_proto_port'] != -1))):
                              flowPriority = 103
                        elif(datax['proto'] != -1):
                              flowPriority = 102
                        else:
                              flowPriority = 101
                              wildcardProtoAndPort = True
```

```
                        pFlowMatch = parser.OFPMatch(**dtFlowMatch)
                        pFlowAction = []

      pFlowAction.append(parser.OFPActionOutput(datax['out_port'], 0))
                        self.addFlow(self.mSwitch[switchId], 0,
flowPriority, pFlowMatch, pFlowAction)

                        if wildcardProtoAndPort:
                              #add for ARP
                              flowPriority = 100
                              dtFlowMatch = {}
                              dtFlowMatch['eth_type'] = 0x0806
                              dtFlowMatch['arp_spa'] = datax['src_ipv4']
                              dtFlowMatch['arp_tpa'] = datax['dst_ipv4']
                              pFlowMatch = parser.OFPMatch(**dtFlowMatch)
                              pFlowAction = []

      pFlowAction.append(parser.OFPActionOutput(datax['out_port'], 0))
                              self.addFlow(self.mSwitch[switchId], 0,
flowPriority, pFlowMatch, pFlowAction)

            return True

      def applyFlowTableBySegment(self, pSwitchId, pSegmentId,
pExprTime):
            pSegmentId = pSegmentId.replace('%20','/')
            pSegmentId = pSegmentId.replace(' ','/')
            self.mLogger.info('start apply flow table by segment
'+pSegmentId+' with expr '+str(pExprTime)+' ...')
            self.mLogger.info('load switch '+pSwitchId+'...')
            arSwitchId = []
            if pSwitchId == 'ALL':
                  for i in self.mSwitch:
                        arSwitchId.append(i)
            else:
                  if int(pSwitchId) in self.mSwitch:
                        arSwitchId.append(int(pSwitchId))

            for switchId in arSwitchId:
                  #self.emptyFlowTable(self.mSwitch[switchId], 0)
                  ofproto = self.mSwitch[switchId].ofproto
                  parser = self.mSwitch[switchId].ofproto_parser

                  strSql = 'select * from `t_flow_table` where '+ \
                                    '`switch_id`="'+str(switchId)+'" and
'+ \
                                    '`segment_id`="'+pSegmentId+'"'
                  dtFlowTable1 = self.mDatabase.getArrayResult(strSql)
                  #print strSql
                  #print dtFlowTable1
                  for datax in dtFlowTable1:
                        dtFlowMatch = {}
                        dtFlowMatch['eth_type'] = 0x0800
                        dtFlowMatch['ipv4_src'] = datax['src_ipv4']
                        dtFlowMatch['ipv4_dst'] = datax['dst_ipv4']
                        if(datax['proto'] != -1):
                              dtFlowMatch['ip_proto'] = datax['proto']
                        if(datax['src_proto_port'] != -1):
                              if(datax['proto'] == 6):
```

```
                                    dtFlowMatch['tcp_src'] =
datax['src_proto_port']
                              elif(datax['proto'] == 17):
                                    dtFlowMatch['udp_src'] =
datax['src_proto_port']
                              elif(datax['proto'] == 132):
                                    dtFlowMatch['sctp_src'] =
datax['src_proto_port']
                        if(datax['dst_proto_port'] != -1):
                              if(datax['proto'] == 6):
                                    dtFlowMatch['tcp_dst'] =
datax['dst_proto_port']
                              elif(datax['proto'] == 17):
                                    dtFlowMatch['udp_dst'] =
datax['dst_proto_port']
                              elif(datax['proto'] == 132):
                                    dtFlowMatch['sctp_dst'] =
datax['dst_proto_port']

                        #memilih priority
                        wildcardProtoAndPort = False
                        if((datax['proto'] != -1) and
((datax['src_proto_port'] != -1) or (datax['dst_proto_port'] != -1))):
                              flowPriority = 103
                        elif(datax['proto'] != -1):
                              flowPriority = 102
                        else:
                              flowPriority = 101
                              wildcardProtoAndPort = False

                        pFlowMatch = parser.OFPMatch(**dtFlowMatch)
                        pFlowAction = []

      pFlowAction.append(parser.OFPActionOutput(datax['out_port'], 0))
                        self.addFlow(self.mSwitch[switchId], 0,
flowPriority, pFlowMatch, pFlowAction)

                        if wildcardProtoAndPort:
                              #add for ARP
                              flowPriority = 100
                              dtFlowMatch = {}
                              dtFlowMatch['eth_type'] = 0x0806
                              dtFlowMatch['arp_spa'] = datax['src_ipv4']
                              dtFlowMatch['arp_tpa'] = datax['dst_ipv4']
                              pFlowMatch = parser.OFPMatch(**dtFlowMatch)
                              pFlowAction = []

      pFlowAction.append(parser.OFPActionOutput(datax['out_port'], 0))
                              self.addFlow(self.mSwitch[switchId], 0,
flowPriority, pFlowMatch, pFlowAction)

            return True

      def emptyFlowTable2(self, pSwitchId):
            self.mLogger.info('start apply flow table from database...')
            self.mLogger.info('load switch '+pSwitchId+'...')
            arSwitchId = []
            if pSwitchId == 'ALL':
                  for i in self.mSwitch:
```

```
                        arSwitchId.append(i)
            else:
                  if int(pSwitchId) in self.mSwitch:
                        arSwitchId.append(int(pSwitchId))

            for switchId in arSwitchId:
                  self.emptyFlowTable(self.mSwitch[switchId],0)

            return True

class MyDynRouteController(ControllerBase):

      def __init__(self, req, link, data, **config):
            super(MyDynRouteController, self).__init__(req, link, data,
**config)
            self.MyDynRouteApp = data[MyDynRouteAppInstanceName]
            self.mLogger = self.MyDynRouteApp.mLogger

      @route('controller', MyDynRouteAppBaseUrl+'discoveryNetwork')
      def discoveryNetwork(self, req, **kwargs):
            self.mLogger.info('wsgi '+str(req))
            dtReply = {}
            dtReply = self.MyDynRouteApp.discoveryNetwork()
            body =json.dumps(dtReply)
            return Response(content_type='application/json', body=body)

      @route('controller', MyDynRouteAppBaseUrl+'getGraphData')
      def getGraphData(self, req, **kwargs):
            self.mLogger.info('wsgi '+str(req))
            dtReply = {}
            dtReply =
json_graph.node_link_data(self.MyDynRouteApp.mGraph)
            body =json.dumps(dtReply)
            return Response(content_type='application/json', body=body)

      @route('controller',
MyDynRouteAppBaseUrl+'applyFlowTableFromDb/{switchId}')
      def applyFlowTableFromDb(self, req, **kwargs):
            self.mLogger.info('wsgi '+str(req))
            dtReply = {}
            dtReply['status'] =
self.MyDynRouteApp.applyFlowTableFromDb(kwargs['switchId'])
            body =json.dumps(dtReply)
            return Response(content_type='application/json', body=body)

      @route('controller',
MyDynRouteAppBaseUrl+'applyFlowTableBySegment/{switchId}/{segmentId}/{exp
rTime}')
      def applyFlowTableBySegment(self, req, **kwargs):
            self.mLogger.info('wsgi '+str(req))
            dtReply = {}
            dtReply['status'] =
self.MyDynRouteApp.applyFlowTableBySegment(kwargs['switchId'],kwargs['seg
mentId'],int(kwargs['exprTime']))
            body =json.dumps(dtReply)
            return Response(content_type='application/json', body=body)

      @route('controller',
MyDynRouteAppBaseUrl+'emptyFlowTable/{switchId}')
```

```
        def emptyFlowTable(self, req, **kwargs):
              self.mLogger.info('wsgi '+str(req))
              dtReply = {}
              dtReply['status'] =
self.MyDynRouteApp.emptyFlowTable2(kwargs['switchId'])
              body =json.dumps(dtReply)
              return Response(content_type='application/json', body=body)
```

Lampiran 3. Script application-plane dsi-app.py

```
#!/usr/bin/python

import signal
import socket
import asyncore
import ConfigParser
import os
import sys
import subprocess
import argparse
import datetime
from datetime import timedelta
import logging
from logging.handlers import TimedRotatingFileHandler
import urllib
import json
import math
from classDB import MyDatabase
import networkx as nx
from networkx import *
from networkx.readwrite import json_graph
import threading
import matplotlib
import matplotlib.pyplot as plt
from wsgiref.simple_server import make_server
from cgi import parse_qs, escape

class MyDsiApp():
      def __init__(self):
            ##
            currentDir = os.path.dirname(os.path.realpath(sys.argv[0]))
            self.mCurrentDir = currentDir
            configFile = currentDir + '/dsi-config.conf'
            self.mConfig = ConfigParser.ConfigParser()
            self.mConfig.read(configFile)

            self.mMininetLayer2Topo =
self.mConfig.get('general','mininet_layer2_topo')
            self.mStartDynPort =
self.mConfig.getint('general','start_dynamic_port')
            self.mRyuUrl = self.mConfig.get('general','ryu_url')
            self.mSflowUrl = self.mConfig.get('general','sflow_url')
            self.mOverloadThreshold =
self.mConfig.getint('overload','load_threshold')
            self.mOverIntervalThreshold =
self.mConfig.getint('overload','over_interval_threshold')
            self.mNormalIntervalThreshold =
self.mConfig.getint('overload','normal_interval_threshold')
            self.mOverloadSflowCsv =
self.mConfig.get('overload','sflow_csv')

            dbHost = self.mConfig.get('database','host')
            dbUser = self.mConfig.get('database','user')
            dbPassword = self.mConfig.get('database','password')
            dbName = self.mConfig.get('database','name')
            logFile = self.mConfig.get('logging','dsi_log_file')
            logLevel = self.mConfig.get('logging','dsi_log_level')
            self.mLogger = logging.getLogger(__name__)
```

```
            logHdl = logging.handlers.TimedRotatingFileHandler(logFile,
when='midnight', backupCount=10)
            logFormatter = logging.Formatter('%(asctime)s %(levelname)s
%(message)s')
            logHdl.setFormatter(logFormatter)
            self.mLogger.addHandler(logHdl)
            if(logLevel == 'CRITICAL'):
                  self.mLogger.setLevel(logging.CRITICAL)
            elif(logLevel == 'ERROR'):
                  self.mLogger.setLevel(logging.ERROR)
            elif(logLevel == 'WARNING'):
                  self.mLogger.setLevel(logging.WARNING)
            elif(logLevel == 'INFO'):
                  self.mLogger.setLevel(logging.INFO)
            elif(logLevel == 'DEBUG'):
                  self.mLogger.setLevel(logging.DEBUG)
            else:
                  self.mLogger.setLevel(logging.DEBUG)

            self.mDatabase = MyDatabase(host=dbHost, user=dbUser,
password=dbPassword, database=dbName, logObj=self.mLogger)
            self.mDatabase.connect()
            self.mDatabase.disconnect()
            strSql = 'update `tbl_setting` set
`val`='+str(self.mOverloadThreshold)+'  where `id`="load_threshold"'
            self.mDatabase.execQuery(strSql)

            #self.mappingOsIntf()
            #self.loadGraph()
            #self.generateDefaultSegment()
            #self.calculateSegmentToFlowTable()

            self.mDataLoadMonitoring = None

            parserCmd = argparse.ArgumentParser()
            parserCmd.add_argument('command', help='Subcommand to run')
            argsCmd = parserCmd.parse_args(sys.argv[1:2])
            if not hasattr(self, argsCmd.command):
                  print 'Unrecognized command'
                  parserCmd.print_help()
                  exit(1)
                  # use dispatch pattern to invoke method with same name

            parserOpt = argparse.ArgumentParser()
            parserOpt.add_argument('--flow-segment-layer', help='flow
segment (proto=3, port_number=4)')
            parserOpt.add_argument('--exclude-links', help='links
excluded in monitoring (ex: ovs1-ovs2,ovs1-ovs5)')
            argsOpt = parserOpt.parse_args(sys.argv[2:])

            if not ((argsOpt.flow_segment_layer
=='3')or(argsOpt.flow_segment_layer =='4')):
                  self.mFlowSegmentLayer = 4
            else:
                  self.mFlowSegmentLayer =
int(argsOpt.flow_segment_layer)

            self.mExcludeIntfName = []
            self.mExcludeIntfOsIndex = []
```

```
            if(argsOpt.exclude_links != None):
                  x1 = argsOpt.exclude_links.strip()
                  x2 = x1.split(',')
                  for x3 in x2:
                        x4 = x3.split('-')
                        strSql = 'select * from `v_link` where
`src_node_label`="'+x4[0]+'" and `dst_node_label`="'+x4[1]+'"'
                        x5 = self.mDatabase.getArrayResult(strSql)
                        for x6 in x5:

      self.mExcludeIntfName.append(x6['src_intf_os_name'])

      self.mExcludeIntfName.append(x6['dst_intf_os_name'])

      self.mExcludeIntfOsIndex.append(x6['src_intf_os_index'])

      self.mExcludeIntfOsIndex.append(x6['dst_intf_os_index'])

            strExcludeIntf = ','.join(self.mExcludeIntfName)
            strSql = 'update `tbl_setting` set `val`="'+strExcludeIntf+'"
where `id`="exclude_intf"'
            self.mDatabase.execQuery(strSql)

            getattr(self, argsCmd.command)()

      def doPreparationProcess(self):
            dtResult = {}
            try:
                  x1 =
self.loadJsonFromUrl(self.mRyuUrl+'discoveryNetwork')
                  dtResult['networkDiscovery'] = 'SUCCESS'
            except:
                  dtResult['networkDiscovery'] = 'ERROR'
            dtResult['mappingOsIntf'] = self.mappingOsIntf()
            dtResult['updateLinkBw'] = self.updateLinkBw()
            dtResult2 = self.resetFlowTable(False)
            dtResult.update(dtResult2)
            print json.dumps(dtResult, sort_keys=True, indent=4,
separators=(',', ': '))

      def resetFlowTable(self, printResult = True):
            dtResult = {}
            dtResult['generateDefaultSegment'] =
self.generateDefaultSegment()
            dtResult['calculateSegmentToFlowTable'] =
self.calculateSegmentToFlowTable()
            dtResult['applyFlowTableFromDb'] =
urllib.urlopen(self.mRyuUrl+'applyFlowTableFromDb/ALL').read()

            if printResult:
                  print json.dumps(dtResult, sort_keys=True, indent=4,
separators=(',', ': '))
            return dtResult

      def loadJsonFromUrl(self, url):
            output = {}
            try:
                  data = urllib.urlopen(url).read()
                  output = json.loads(data)
```

```
            except:
                  pass
            return output

      def mappingOsIntf(self):
            while(self.mDatabase.getSingleResult('SELECT COUNT(*) FROM
`v_intf` WHERE `dpid` IS NOT NULL AND `os_index` IS NULL')>0):
                  sflowAgents =
self.loadJsonFromUrl(self.mSflowUrl+'agents/json')
                  for agent in sflowAgents:
                        #print agent
                        metricAgent =
self.loadJsonFromUrl(self.mSflowUrl+'metric/'+agent+'/json')
                        for key in metricAgent:
                              x1 = key.split('.')
                              if(len(x1) >=2 ):
                                    if(x1[1] == 'of_port'):
                                          dpid =
int(metricAgent[x1[0]+'.of_dpid'], 16)
                                          of_port =
int(metricAgent[x1[0]+'.of_port'])
                                          os_index =
int(metricAgent[x1[0]+'.ifindex'])
                                          os_name =
metricAgent[x1[0]+'.ifname']
                                          strSql = 'update `t_intf` set
'+ \
      ''`os_index`='+str(os_index)+','+ \
      ''`os_name`="'+os_name+'" '+ \
                                                ' where '+ \
      ''`dpid`='+str(dpid)+' and '+ \
      ''`port_no`='+str(of_port)
      self.mDatabase.execQuery(strSql)
                                          if(of_port == 65534):
                                                strSql = 'update
`t_switch` set '+ \
      ''`sflow_agent`="'+agent+'",'+ \
      ''`label`="'+os_name+'" '+ \
                                                      ' where
`dpid`="'+str(dpid)+'"'
      self.mDatabase.execQuery(strSql)

            #print 'SUCCESS'
            return 'SUCCESS'

      def updateLinkBw(self):
            file = open(self.mMininetLayer2Topo,'r')
            for line in file:
                  node1 = None
                  node2 = None
                  bw = None
```

```
                if(('net.addLink' in line) and \
                      ('cls=TCLink' in line) and \
                      ('bw=' in line)):
                      x1 = line.split('(');
                      if(len(x1)>= 2):
                            x2 = x1[1].split(')')
                            if(len(x2)>= 1):
                                  x3 = x2[0].split(',')
                                  if(len(x3) >= 4):
                                        node1 = x3[0].strip()
                                        node2 = x3[1].strip()
                                        for x4 in x3:
                                              if('bw=' in x4):
                                                    x5 = x4.split('=')
                                                    #print x5
                                                    if(len(x5) >= 2):
                                                          bw =
int(x5[1].strip())
                if((node1 is not None)and(node2 is not None)and(bw is
not None)):
                      strSql = 'select * from `v_link` where '+ \
                                                    '(`src_node_label`="'+node1+'"
and '+ \
                                                    '`dst_node_label`="'+node2+'")
or '+ \
                                                    '(`src_node_label`="'+node2+'"
and '+ \
                                                    '`dst_node_label`="'+node1
      +'")'
                      ar1 = self.mDatabase.getArrayResult(strSql)
                      for pRecord in ar1:
                            strSql = 'update `t_link` '+ \
                                                    ' set '+ \
                                                    '`bw`='+str(bw*1000000)+'
'+ \
                                                    ' where '+ \
      '`src_intf_id`="'+pRecord['src_intf_id']+'" and '+ \
      '`dst_intf_id`="'+pRecord['dst_intf_id']+'"'
                                  #print strSql
                                  self.mDatabase.execQuery(strSql)
            #print 'SUCCESS'
            return 'SUCCESS'

      def loadGraph(self):
            self.mLogger.info('load data graph')
            pGraph = nx.DiGraph()
            ar1 = self.mDatabase.getArrayResult('select * from `v_node`')
            for pRecord in ar1:
                  if not pRecord['id'] in pGraph.node:
                        pGraph.add_node(pRecord['id'])
                        pGraph.node[pRecord['id']]['data'] = pRecord
                        pGraph.node[pRecord['id']]['intf'] = {}

            ar1 = self.mDatabase.getArrayResult('select * from `v_intf`')
            for pRecord in ar1:
                  if pRecord['node_id'] in pGraph.node:
```

```
		pGraph.node[pRecord['node_id']]['intf'][pRecord['id']] = pRecord

			ar1 = self.mDatabase.getArrayResult('select * from `v_link`')
			for pRecord in ar1:
				#pWeight = self.mMaxLinkBw / pRecord['bw']
				pWeight = 0
				#print 'Add edge
'+pRecord['device_id_src']+'==>'+pRecord['device_id_dst']

		pGraph.add_edge(pRecord['src_node_id'],pRecord['dst_node_id'],
weight=pWeight, \
									load=0, data=pRecord)

			#self.saveGraphToImageFile(pGraph)
			#data = json_graph.node_link_data(pGraph)
			#print json.dumps(data, sort_keys=True, indent=4,
separators=(',', ': '))
			return pGraph

		def saveGraphToImageFile(self, pGraph):
			pos=nx.spectral_layout(pGraph)
			labels={}
			for xnode in pGraph.nodes():
				labels[xnode]=pGraph.node[xnode]['data']['label']
				#labels[xnode]=xnode

			#nx.draw(pGraph)
			nx.draw_networkx_nodes(pGraph,pos)
			nx.draw_networkx_edges(pGraph,pos,width=1.0,alpha=0.5)
			nx.draw_networkx_labels(pGraph,pos,labels)
			plt.axis('off')
			#plt.savefig("simple_path.png")
			plt.show()
			pass

		def generateDefaultSegment(self, isEmptyExisting = True):
			self.mLogger.info('generate default segment')
			self.mGraph = self.loadGraph()
			csvFile = self.mCurrentDir+'/tmp/defaultSegmen.csv'
			fileHdl = open(csvFile, 'w')
			if isEmptyExisting:
				self.mDatabase.execQuery('delete from
`t_traffic_segment`')
			pGraph = self.loadGraph()
			for pNode1 in pGraph.nodes():
				#print pGraph.node[pNode1]
				for pNode2 in pGraph.nodes():
					if (
							(pNode1!=pNode2) and
							(pGraph.node[pNode1]['data']['type']
== 'HOST') and
							(pGraph.node[pNode2]['data']['type']
== 'HOST')
						):
						srcIp = pGraph.node[pNode1]['data']['ipv4']
						dstIp = pGraph.node[pNode2]['data']['ipv4']
						segmentId = srcIp+'/-1/-1/-1/'+dstIp
						csvLine = segmentId+"\t"+ \
```

```
                                                srcIp+"\t"+ \
                                                '-1'+"\t"+ \
                                                '-1'+"\t"+ \
                                                '-1'+"\t"+ \
                                                dstIp+"\t"+ \
                                                '1'+"\n"
                                    fileHdl.write(csvLine)
                  fileHdl.close()
                  strSql = 'load data local infile "'+csvFile+'" into table
`t_traffic_segment` ('+ \

            '`id`,`src_ipv4`,`src_proto_port`,`proto`,`dst_proto_port`,`dst_ipv
4`,`load`'+ \
                                    ') '
                  self.mDatabase.execQuery(strSql)
                  #print 'SUCCESS'
                  return 'SUCCESS'

            def calculateSegmentToFlowTable(self):
                  self.mLogger.info('calculate segment to flow routing')
                  self.mDatabase.execQuery('delete from `t_flow_table`')
                  self.mGraph = self.loadGraph()
                  flowTableId = 1
                  arSegment = self.mDatabase.getArrayResult('select * from
`t_traffic_segment` order by `load` desc')

                  csvFile = self.mCurrentDir+'/tmp/flowTable.csv'
                  fileHdl = open(csvFile, 'w')
                  #print arSegment
                  for pSegment in arSegment:
                        srcIpv4 = pSegment['src_ipv4']
                        dstIpv4 = pSegment['dst_ipv4']
                        srcNodeId = self.mDatabase.getSingleResult('select
`hw_addr` from `t_host` where `ipv4`="'+srcIpv4+'"')
                        dstNodeId = self.mDatabase.getSingleResult('select
`hw_addr` from `t_host` where `ipv4`="'+dstIpv4+'"')
                        proto = pSegment['proto']
                        if (pSegment['src_proto_port'] < self.mStartDynPort):
                              srcProtoPort = pSegment['src_proto_port']
                        else:
                              srcProtoPort = -1
                        if (pSegment['dst_proto_port'] < self.mStartDynPort):
                              dstProtoPort = pSegment['dst_proto_port']
                        else:
                              dstProtoPort = -1
                        pRouteNode = None
                        try:
                              pRouteNode =
nx.dijkstra_path(self.mGraph,srcNodeId,dstNodeId)
                        except:
                              pass
                        #print srcIpv4
                        #print dstIpv4
                        #print 'select `hw_addr` from `t_host` where
`ipv4`="'+srcIpv4+'"'
                        #print srcNodeId
                        #print dstNodeId
                        #print pRouteNode
                        if pRouteNode is not None:
```

```
                        for i in range(1, len(pRouteNode)-1):
                              srcEdgeNodeId = pRouteNode[i]
                              dstEdgeNodeId = pRouteNode[i+1]
                              dtNextEdge =
self.mGraph.get_edge_data(srcEdgeNodeId, dstEdgeNodeId)
                              #print dtNextEdge
                              outPort =
dtNextEdge['data']['src_intf_port_no']
                              csvLine = str(flowTableId)+"\t"+ \
                                          pSegment['id']+"\t"+ \
                                          str(i)+"\t"+ \
                                          str(srcEdgeNodeId)+"\t"+ \
                                          srcIpv4+"\t"+ \
                                          dstIpv4+"\t"+ \
                                          str(proto)+"\t"+ \
                                          str(srcProtoPort)+"\t"+ \
                                          str(dstProtoPort)+"\t"+ \
                                          str(outPort)+"\n"
                              #print csvLine
                              fileHdl.write(csvLine)
                              flowTableId = flowTableId + 1
                              loadRatio = 1

                              linkBw =
self.mGraph[pRouteNode[i]][pRouteNode[i+1]]['data']['bw']
                              if linkBw>0:
                                    weightBefore =
self.mGraph[pRouteNode[i]][pRouteNode[i+1]]['weight']
                                    loadBefore =
self.mGraph[pRouteNode[i]][pRouteNode[i+1]]['load']
                                    loadAfter = loadBefore +
((pSegment['load'] / loadRatio)/(linkBw/10000000))
                                    #weightAfter = loadAfter *
(self.mMaxLinkBw / linkBw) #mempertimbangkan bw link
                                    weightAfter = loadAfter       #hanya
mempertimbangkan load saja
                              else:
                                    loadAfter = 0
                                    weightAfter = 0

                              if(dtNextEdge['data']['dst_intf_os_name']
in self.mExcludeIntfName):
                                    loadAfter = 0
                                    weightAfter = 0


      self.mGraph[pRouteNode[i]][pRouteNode[i+1]]['load']= loadAfter

      self.mGraph[pRouteNode[i]][pRouteNode[i+1]]['weight']= weightAfter

                              #print linkBw
                              #print weightBefore
                              #print loadBefore
                              #print loadAfter
                              #print weightAfter

            fileHdl.close()
            strSql = 'load data local infile "'+csvFile+'"  into table
`t_flow_table` ('+ \
```

```
      '`id`,`segment_id`,`hop_index`,`switch_id`,`src_ipv4`,`dst_ipv4`,`p
roto`,'+ \
                        '`src_proto_port`,`dst_proto_port`,`out_port`'+ \
                        ')'
            #print strSql
            self.mDatabase.execQuery(strSql)
            #print 'SUCCESS'
            return 'SUCCESS'

      def startBwmNg(self):
            strSql = 'SELECT `src_intf_name` FROM `v_link` WHERE
`src_node_type`="SWITCH" AND `dst_node_type`="SWITCH"'
            x1 = self.mDatabase.getArrayResult(strSql)
            strIntf = ''
            for x2 in x1:
                  strIntf = strIntf + x2['src_intf_name'] + ','
            if(strIntf != ''): strIntf = strIntf[:-1]
            #print strIntf
            dtResult = {}
            strTimeStamp =
datetime.datetime.now().strftime("%Y%m%d%H%M%S")

            csvFile1 = self.mMonitoringCsvDir+'/bwm-ng-serial-
'+strTimeStamp+'.csv'
            strCmd1 = '/usr/bin/bwm-ng -t 1000 -I '+strIntf+' -T avg -t
10000 -A 21 -o csv -F '+csvFile1+' &'
            #print strCmd1
            return
            process1 = subprocess.call(strCmd1, shell=True)
            if process1 == 0:
                  print 'SUCCESS'

      def checkBwmNg(self):
            strCmd1 = 'ps ax | grep bwm-ng'
            process1 = subprocess.call(strCmd1, shell=True)
            if process1 ==0:
                  print 'SUCCESS'

      def stopBwmNg(self):
            strCmd1 = '/bin/ps a'
            process1 = subprocess.Popen(strCmd1.split(' '),
stdout=subprocess.PIPE, stderr=subprocess.PIPE)
            out, err = process1.communicate()
            x1 = out.split("\n")
            for x2 in x1:
                  if(
                        (x2.find('bwm-ng') >= 0)and
                        (x2.find('eth') >= 0) and
                        (x2.find(',') >= 0)
                  ):
                        #print '====='
                        #print x2
                        x3 = x2.strip().split(' ')[0]
                        #print x3
                        strCmd2 = 'kill -9 '+x3
                        process2 = subprocess.call(strCmd2, shell=True)

            #if process1 ==0:
```

```
            print 'SUCCESS'

      def doMonitoringPathLoad(self):
            #self.mMonitoringCurrentBwCsv
            if(self.mDataLoadMonitoring is None):
                  self.mLogger.info('Using flow segment layer
'+str(self.mFlowSegmentLayer))
                  self.mLogger.info('Start monitoring path load...')
                  print 'Using flow segment layer
'+str(self.mFlowSegmentLayer)
                  print 'Start monitoring overload...'

                  self.mDataLoadMonitoring = {}
                  strSql = 'SELECT * FROM `v_link` WHERE
`src_node_type`="SWITCH" AND `dst_node_type`="SWITCH"'
                  x1 = self.mDatabase.getArrayResult(strSql)
                  for x2 in x1:
                        if(not x2['dst_intf_name'] in
self.mExcludeIntfName):

      self.mDataLoadMonitoring[x2['dst_intf_name']] = {

                                    'load' : 0,

                                    'threshold' : x2['bw'] *
self.mOverloadThreshold / 100,

                                    'flag_state' : 'NORMAL',

                                    'flag_overload_time' : 0,

                                    'flag_normal_time' : 0

                              }
                  #print self.mDataLoadMonitoring
                  if os.path.exists(self.mOverloadSflowCsv):
                        os.remove(self.mOverloadSflowCsv)
                  strSql = 'delete from `t_overload_flow`'
                  self.mDatabase.execQuery(strSql)

            pNeedOverloadMonitoring = False
            #strCmd = 'bwm-ng -T avg -o csv -c 1 -T avg -t -A 21 >
/var/log/dsi/csv/bwm-ng-current.csv'
            strCmd = 'bwm-ng -o csv -c 1 > /var/log/dsi/csv/bwm-ng-
current.csv'
            process = subprocess.Popen(strCmd.split(' '),
stdout=subprocess.PIPE, stderr=subprocess.PIPE)
            out, err = process.communicate()
            #print out
            x1 = out.split("\n")
            for x2 in x1:
                  #print x2
                  x3 = x2.split(';')
                  if(len(x3)>=10):
                        #print x3[1]
                        #if((x3[1] in self.mDataLoadMonitoring) and
(x3[1] == 'ovs1-eth2')):
                        if((x3[1] in self.mDataLoadMonitoring)):
                              intfName = x3[1]
```

```
                                    #save to db
                                    simulateFromDb = False
                                    if not simulateFromDb:
                                          bytes_out_s =
int(x3[2].split('.')[0])
                                          bytes_in_s = int(x3[3].split('.')[0])
                                          strSql = 'update `t_intf` set
bytes_out_s='+str(bytes_out_s)+', bytes_in_s='+str(bytes_in_s)+' where
`name`="'+intfName+'"'
                                          self.mDatabase.execQuery(strSql)
                                          #print strSql
                                    else:
                                          strSql = 'select * from `t_intf`
where `name`="'+intfName+'"'
                                          x4 =
self.mDatabase.getArrayResult(strSql)[0]
                                          bytes_out_s = x4['bytes_out_s']
                                          bytes_in_s = x4['bytes_in_s']

                                    prevLoad =
self.mDataLoadMonitoring[intfName]['load']
                                    currentLoad = int(bytes_in_s) * 8
                                    #print '------------'
                                    #print currentLoad

      if(self.mDataLoadMonitoring[intfName]['flag_state'] == 'NORMAL'):
                                          #deteksi OVERLOAD pakai threshold
                                          if(currentLoad >
self.mDataLoadMonitoring[intfName]['threshold']):

      self.mDataLoadMonitoring[intfName]['flag_overload_time'] =
datetime.datetime.now()

      self.mDataLoadMonitoring[intfName]['flag_normal_time'] = 0

      self.mDataLoadMonitoring[intfName]['flag_state'] = 'OVERLOAD'

                                                self.mLogger.info('start record
overload flow...')
                                                print intfName+' OVERLOAD, wait
until '+str(self.mOverIntervalThreshold)+' seconds...'
                                    else:
                                          pNeedOverloadMonitoring = True

                                          #deteksi NORMAL pakai threshold
                                          if(currentLoad <=
self.mDataLoadMonitoring[intfName]['threshold']):
                                                if(prevLoad >
self.mDataLoadMonitoring[intfName]['threshold']):

      self.mDataLoadMonitoring[intfName]['flag_normal_time'] =
datetime.datetime.now()
                                                      print intfName+'
OVERLOAD, detect NORMAL, wait until
'+str(self.mNormalIntervalThreshold)+' seconds to realy switch to NORMAL'
                                          else:

      if(self.mDataLoadMonitoring[intfName]['flag_normal_time'] != 0):
```

```
                self.mDataLoadMonitoring[intfName]['flag_normal_time'] = 0
                                                print intfName+'
OVERLOAD, previous detect NORMAL, before
'+str(self.mNormalIntervalThreshold)+' back to OVERLOAD again'
                            #deteksi jika kembali ke state NORMAL
dengan mNormalIntervalThreshold
                            if    (

                self.mDataLoadMonitoring[intfName]['flag_normal_time'] != 0) and

                (self.mDataLoadMonitoring[intfName]['flag_normal_time'] >
self.mDataLoadMonitoring[intfName]['flag_overload_time']) and
                                                ((datetime.datetime.now()
- self.mDataLoadMonitoring[intfName]['flag_normal_time']).total_seconds()
> self.mNormalIntervalThreshold)
                                        ):

                self.mDataLoadMonitoring[intfName]['flag_overload_time'] = 0

                self.mDataLoadMonitoring[intfName]['flag_normal_time'] =
datetime.datetime.now()

                self.mDataLoadMonitoring[intfName]['flag_state'] = 'NORMAL'
                                                print intfName+' NORMAL, after
'+str(self.mNormalIntervalThreshold)+' seconds consist detect NORMAL'

                            #deteksi jika tetap state OVERLOAD
sampai batas mOverIntervalThreshold
                            if    (

                (self.mDataLoadMonitoring[intfName]['flag_state'] == 'OVERLOAD')
and
                                                ((datetime.datetime.now()
-
self.mDataLoadMonitoring[intfName]['flag_overload_time']).total_seconds()
> self.mOverIntervalThreshold)
                                        ):

                if(self.mDataLoadMonitoring[intfName]['flag_normal_time'] != 0):
                                                        print intfName+'
OVERLOAD, after '+str(self.mOverIntervalThreshold)+' seconds, but still
detect NORMAL, wait detect NORMAL consist for
'+str(self.mNormalIntervalThreshold)+' seconds'
                                                else:

                pNeedOverloadMonitoring = False
                                                                print intfName+'
RECALCULATE TRIGERED'
                                                                print
'calculateOverloadFlowToSegment...'

                self.calculateOverloadFlowToSegment()
                                                                print
'calculateSegmentToFlowTable...'

                self.calculateSegmentToFlowTable()
                                                                print
'applyFlowTableFromDb...'
```

```
        urllib.urlopen(self.mRyuUrl+'applyFlowTableFromDb/ALL').read()
                                                        print 'SUCCESS'

        self.mDataLoadMonitoring = None
                                                        break

                                    self.mDataLoadMonitoring[intfName]['load']
= currentLoad
                if pNeedOverloadMonitoring:
                        self.doMonitoringOverloadFlow()
                self.mMonitoringInterval =
self.mConfig.getint('general','monitoring_interval')
                self.timerMonitoringOverload =
threading.Timer(self.mMonitoringInterval, self.doMonitoringPathLoad)
                self.timerMonitoringOverload.start()

        def doMonitoringOverloadFlow(self):
                strTimeStamp = datetime.datetime.now().strftime('%Y-%m-%d
%H:%M:%S')
                print 'doMonitoringOverloadFlow '+strTimeStamp
                dtFlows =
self.loadJsonFromUrl(self.mSflowUrl+'activeflows/ALL/mn_flow/json?aggMode
=avg')
                #print
self.mSflowUrl+'activeflows/ALL/mn_flow/json?aggMode=avg'
                #print dtFlows
                fileHdl = open(self.mOverloadSflowCsv, 'a+')
                for dtFlow in dtFlows:
                        load = dtFlow['value']
                        x1 = dtFlow['key']
                        x2 = x1.split('_SEP_')
                        srcIpv4 = x2[0]
                        dstIpv4 = x2[1]
                        proto = int(x2[2])
                        srcProtoPort = int(x2[3])
                        if(srcProtoPort >= self.mStartDynPort):
                                srcProtoPort = -1
                        dstProtoPort = int(x2[4])

                        if(dstProtoPort >= self.mStartDynPort):
                                dstProtoPort = -1
                        segmentId1 = srcIpv4+'/'+ \
                                     str(srcProtoPort)+'/'+ \
                                     str(proto)+'/'+ \
                                     str(dstProtoPort)+'/'+ \
                                     dstIpv4
                        segmentId2 = srcIpv4+'/'+ \
                                     '-1/'+ \
                                     str(proto)+'/'+ \
                                     '-1/'+ \
                                     dstIpv4
                        csvLine = strTimeStamp+"\t"+ \
                                     segmentId1+"\t"+ \
                                     segmentId2+"\t"+ \
                                     srcIpv4+"\t"+ \
                                     str(srcProtoPort)+"\t"+ \
                                     str(proto)+"\t"+ \
                                     str(dstProtoPort)+"\t"+ \
```

```
                              dstIpv4+"\n"
                        #print csvLine
                        fileHdl.write(csvLine)
                        strSql = 'insert into `t_overload_flow` '+ \

      '(`time`,`segment_id_1`,`segment_id_2`,`src_ipv4`,`src_proto_port`,
`proto`,'+ \
                              ``dst_proto_port`,`dst_ipv4`,`load`) values
('+ \
                              '"'+strTimeStamp+'",'+ \
                              '"'+segmentId1+'",'+ \
                              '"'+segmentId2+'",'+ \
                              '"'+srcIpv4+'",'+ \
                              str(srcProtoPort)+','+ \
                              str(proto)+','+ \
                              str(dstProtoPort)+','+ \
                              '"'+dstIpv4+'",'+ \
                              str(load)+ \
                              ') on duplicate key update
`load`=(`load`+'+str(load)+')'
                        #print strSql
                        self.mDatabase.execQuery(strSql)
                  fileHdl.close()

      def calculateOverloadFlowToSegment(self):
                  self.mLogger.info('calculate load to segment')
                  self.mMinSegmentLoad =
int(self.mConfig.get('general','min_segment_load'))
                  self.generateDefaultSegment();
                  csvFile = self.mCurrentDir+'/tmp/flowLoadToSegmen.csv'
                  fileHdl = open(csvFile, 'w')

                  #utk flow load yg >= self.mMinSegmentLoad
                  #strSql = 'select * from `t_overload_flow` where
`load`>='+str(self.mMinSegmentLoad)
                  if self.mFlowSegmentLayer == 4:
                        strSql = 'SELECT *, AVG(`load`)AS `load_avg` FROM
`t_overload_flow` '+ \
                              'WHERE `load`>='+str(self.mMinSegmentLoad)+'
GROUP BY `segment_id_1`'
                  elif self.mFlowSegmentLayer == 3:
                        strSql = 'SELECT *, AVG(`load`)AS `load_avg` FROM
`t_overload_flow` '+ \
                              'WHERE `load`>='+str(self.mMinSegmentLoad)+'
GROUP BY `proto`'

                  #print strSql
                  recordFlowLoad = self.mDatabase.getArrayResult(strSql);
                  for dtFlowLoad in recordFlowLoad:
                        if self.mFlowSegmentLayer == 4:
                              segmentId = dtFlowLoad['segment_id_1']
                        elif self.mFlowSegmentLayer == 3:
                              segmentId = dtFlowLoad['segment_id_2']

                        csvLine = segmentId+"\t"+ \
                                    dtFlowLoad['src_ipv4']+"\t"+ \

      str(dtFlowLoad['src_proto_port'])+"\t"+ \
                                    str(dtFlowLoad['proto'])+"\t"+ \
```

```
        str(dtFlowLoad['dst_proto_port'])+"\t"+ \
                                        dtFlowLoad['dst_ipv4']+"\t"+ \
                                        str(dtFlowLoad['load_avg'])+"\n"
                    #print csvLine
                    fileHdl.write(csvLine)

                fileHdl.close()
                strSql = 'load data local infile "'+csvFile+'" into table
`t_traffic_segment` ('+ \

        '`id`,`src_ipv4`,`src_proto_port`,`proto`,`dst_proto_port`,`dst_ipv
4`,`load`'+ \
                                    ') '
                self.mDatabase.execQuery(strSql)
                #print 'SUCCESS'
                return 'SUCCESS'

if __name__ == "__main__":
        MyDsiApp()
```

Lampiran 4. Database sqldump

```
/*
SQLyog Ultimate v11.33 (64 bit)
MySQL - 10.0.31-MariaDB-0ubuntu0.16.04.2 : Database - db_tiram_dsi
*********************************************************************
*/
/*!40101 SET NAMES utf8 */;
/*!40101 SET SQL_MODE=''*/;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO'
*/;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;
CREATE DATABASE /*!32312 IF NOT EXISTS*/`db_tiram_dsi` /*!40100 DEFAULT
CHARACTER SET utf8mb4 */;
/*Table structure for table `t_flow_table` */
CREATE TABLE `t_flow_table` (
  `id` int(11) NOT NULL,
  `segment_id` varchar(100) DEFAULT NULL,
  `hop_index` int(11) DEFAULT NULL,
  `switch_id` varchar(11) DEFAULT NULL,
  `src_ipv4` varchar(20) DEFAULT NULL,
  `dst_ipv4` varchar(20) DEFAULT NULL,
  `proto` int(11) DEFAULT NULL,
  `src_proto_port` int(11) DEFAULT NULL,
  `dst_proto_port` int(11) DEFAULT NULL,
  `out_port` int(11) DEFAULT NULL,
  `priority` int(11) DEFAULT '0',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
/*Table structure for table `t_host` */
CREATE TABLE `t_host` (
  `id` varchar(30) NOT NULL,
  `hw_addr` varchar(30) DEFAULT NULL,
  `ipv4` varchar(30) DEFAULT NULL,
  `label` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
/*Table structure for table `t_intf` */
CREATE TABLE `t_intf` (
  `id` varchar(30) NOT NULL,
  `hw_addr` varchar(30) DEFAULT NULL,
  `name` varchar(30) DEFAULT NULL,
  `dpid` bigint(30) DEFAULT NULL,
  `port_no` bigint(20) DEFAULT NULL,
  `os_index` int(11) DEFAULT NULL,
  `os_name` varchar(100) DEFAULT NULL,
  `bytes_in_s` bigint(20) DEFAULT '0',
  `bytes_out_s` bigint(20) DEFAULT '0',
  `bytes_in_udp` bigint(20) DEFAULT '0',
  `tcp_flow_count` bigint(20) DEFAULT '0',
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
/*Table structure for table `t_link` */
CREATE TABLE `t_link` (
  `src_intf_id` varchar(50) NOT NULL,
  `dst_intf_id` varchar(50) NOT NULL,
  `bw` bigint(20) DEFAULT NULL,
  PRIMARY KEY (`src_intf_id`,`dst_intf_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

```
/*Table structure for table `t_overload_flow` */
CREATE TABLE `t_overload_flow` (
  `time` datetime NOT NULL,
  `segment_id_1` varchar(100) DEFAULT NULL,
  `segment_id_2` varchar(100) DEFAULT NULL,
  `src_ipv4` varchar(20) NOT NULL,
  `dst_ipv4` varchar(20) NOT NULL,
  `proto` int(11) NOT NULL,
  `src_proto_port` int(11) NOT NULL,
  `dst_proto_port` int(11) NOT NULL,
  `load` bigint(11) DEFAULT NULL,
  PRIMARY KEY
(`time`,`src_ipv4`,`dst_ipv4`,`proto`,`src_proto_port`,`dst_proto_port`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
/*Table structure for table `t_switch` */
CREATE TABLE `t_switch` (
  `id` varchar(30) NOT NULL,
  `dpid` bigint(20) DEFAULT NULL,
  `sflow_agent` varchar(100) DEFAULT NULL,
  `label` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
/*Table structure for table `t_traffic_segment` */
CREATE TABLE `t_traffic_segment` (
  `id` varchar(100) NOT NULL,
  `src_ipv4` varchar(20) DEFAULT NULL,
  `src_proto_port` int(11) DEFAULT NULL,
  `proto` int(11) DEFAULT NULL,
  `dst_proto_port` int(11) DEFAULT NULL,
  `dst_ipv4` varchar(20) DEFAULT NULL,
  `load` bigint(20) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
/*Table structure for table `v_flow_table_link` */
DROP TABLE IF EXISTS `v_flow_table_link`;
/*!50001 CREATE TABLE  `v_flow_table_link`(
 `id` int(11) ,
 `segment_id` varchar(100) ,
 `hop_index` int(11) ,
 `switch_id` varchar(11) ,
 `src_ipv4` varchar(20) ,
 `dst_ipv4` varchar(20) ,
 `proto` int(11) ,
 `src_proto_port` int(11) ,
 `dst_proto_port` int(11) ,
 `out_port` int(11) ,
 `src_intf_id` varchar(50) ,
 `dst_intf_id` varchar(50) ,
 `bw` bigint(20) ,
 `src_intf_port_no` bigint(20) ,
 `src_intf_name` varchar(30) ,
 `src_intf_os_index` int(11) ,
 `src_intf_os_name` varchar(100) ,
 `src_node_type` varchar(6) ,
 `src_node_id` varchar(30) ,
 `src_node_dpid` bigint(20) ,
 `src_node_sflow_agent` varchar(100) ,
 `src_node_ipv4` varchar(100) ,
 `src_node_label` varchar(100) ,
```

```
 `dst_intf_port_no` bigint(20) ,
 `dst_intf_name` varchar(30) ,
 `dst_intf_os_index` int(11) ,
 `dst_intf_os_name` varchar(100) ,
 `dst_node_type` varchar(6) ,
 `dst_node_id` varchar(30) ,
 `dst_node_dpid` bigint(20) ,
 `dst_node_sflow_agent` varchar(100) ,
 `dst_node_ipv4` varchar(100) ,
 `dst_node_label` varchar(100)
)*/;
/*Table structure for table `v_intf` */
DROP TABLE IF EXISTS `v_intf`;
/*!50001 CREATE TABLE  `v_intf`(
 `id` varchar(30) ,
 `hw_addr` varchar(30) ,
 `name` varchar(30) ,
 `dpid` bigint(30) ,
 `port_no` bigint(20) ,
 `os_index` int(11) ,
 `os_name` varchar(100) ,
 `bytes_in_s` bigint(20) ,
 `bytes_out_s` bigint(20) ,
 `bytes_in_udp` bigint(20) ,
 `tcp_flow_count` bigint(20) ,
 `node_id` varchar(30)
)*/;
/*Table structure for table `v_link` */
DROP TABLE IF EXISTS `v_link`;
/*!50001 CREATE TABLE  `v_link`(
 `src_intf_id` varchar(50) ,
 `dst_intf_id` varchar(50) ,
 `bw` bigint(20) ,
 `src_intf_port_no` bigint(20) ,
 `src_intf_name` varchar(30) ,
 `src_intf_os_index` int(11) ,
 `src_intf_os_name` varchar(100) ,
 `src_bytes_in_s` bigint(20) ,
 `src_bytes_out_s` bigint(20) ,
 `src_bytes_in_udp` bigint(20) ,
 `src_tcp_flow_count` bigint(20) ,
 `src_node_type` varchar(6) ,
 `src_node_dpid` bigint(20) ,
 `src_node_id` varchar(30) ,
 `src_node_sflow_agent` varchar(100) ,
 `src_node_ipv4` varchar(100) ,
 `src_node_label` varchar(100) ,
 `dst_intf_port_no` bigint(20) ,
 `dst_intf_name` varchar(30) ,
 `dst_intf_os_index` int(11) ,
 `dst_intf_os_name` varchar(100) ,
 `dst_bytes_in_s` bigint(20) ,
 `dst_bytes_out_s` bigint(20) ,
 `dst_bytes_in_udp` bigint(20) ,
 `dst_tcp_flow_count` bigint(20) ,
 `dst_node_type` varchar(6) ,
 `dst_node_dpid` bigint(20) ,
 `dst_node_id` varchar(30) ,
 `dst_node_sflow_agent` varchar(100) ,
```

```
 `dst_node_ipv4` varchar(100) ,
 `dst_node_label` varchar(100)
)*/;
/*Table structure for table `v_node` */
DROP TABLE IF EXISTS `v_node`;
/*!50001 CREATE TABLE  `v_node`(
 `id` varchar(30) ,
 `type` varchar(6) ,
 `hw_addr` varchar(30) ,
 `dpid` bigint(20) ,
 `sflow_agent` varchar(100) ,
 `ipv4` varchar(100) ,
 `label` varchar(100)
)*/;
/*View structure for view v_flow_table_link */
/*!50001 DROP TABLE IF EXISTS `v_flow_table_link` */;
/*!50001 CREATE ALGORITHM=UNDEFINED DEFINER=`tarom`@`%` SQL SECURITY
DEFINER VIEW `v_flow_table_link` AS select `t_flow_table`.`id` AS
`id`,`t_flow_table`.`segment_id` AS
`segment_id`,`t_flow_table`.`hop_index` AS
`hop_index`,`t_flow_table`.`switch_id` AS
`switch_id`,`t_flow_table`.`src_ipv4` AS
`src_ipv4`,`t_flow_table`.`dst_ipv4` AS `dst_ipv4`,`t_flow_table`.`proto`
AS `proto`,`t_flow_table`.`src_proto_port` AS
`src_proto_port`,`t_flow_table`.`dst_proto_port` AS
`dst_proto_port`,`t_flow_table`.`out_port` AS
`out_port`,`v_link`.`src_intf_id` AS `src_intf_id`,`v_link`.`dst_intf_id`
AS `dst_intf_id`,`v_link`.`bw` AS `bw`,`v_link`.`src_intf_port_no` AS
`src_intf_port_no`,`v_link`.`src_intf_name` AS
`src_intf_name`,`v_link`.`src_intf_os_index` AS
`src_intf_os_index`,`v_link`.`src_intf_os_name` AS
`src_intf_os_name`,`v_link`.`src_node_type` AS
`src_node_type`,`v_link`.`src_node_id` AS
`src_node_id`,`v_link`.`src_node_dpid` AS
`src_node_dpid`,`v_link`.`src_node_sflow_agent` AS
`src_node_sflow_agent`,`v_link`.`src_node_ipv4` AS
`src_node_ipv4`,`v_link`.`src_node_label` AS
`src_node_label`,`v_link`.`dst_intf_port_no` AS
`dst_intf_port_no`,`v_link`.`dst_intf_name` AS
`dst_intf_name`,`v_link`.`dst_intf_os_index` AS
`dst_intf_os_index`,`v_link`.`dst_intf_os_name` AS
`dst_intf_os_name`,`v_link`.`dst_node_type` AS
`dst_node_type`,`v_link`.`dst_node_id` AS
`dst_node_id`,`v_link`.`dst_node_dpid` AS
`dst_node_dpid`,`v_link`.`dst_node_sflow_agent` AS
`dst_node_sflow_agent`,`v_link`.`dst_node_ipv4` AS
`dst_node_ipv4`,`v_link`.`dst_node_label` AS `dst_node_label` from
(`t_flow_table` join `v_link` on(((`t_flow_table`.`out_port` =
`v_link`.`src_intf_port_no`) and (`t_flow_table`.`switch_id` =
`v_link`.`src_node_dpid`)))) */;
/*View structure for view v_intf */
/*!50001 DROP TABLE IF EXISTS `v_intf` */;
/*!50001 CREATE ALGORITHM=UNDEFINED DEFINER=`tarom`@`%` SQL SECURITY
DEFINER VIEW `v_intf` AS (select `t_intf`.`id` AS `id`,`t_intf`.`hw_addr`
AS `hw_addr`,`t_intf`.`name` AS `name`,`t_intf`.`dpid` AS
`dpid`,`t_intf`.`port_no` AS `port_no`,`t_intf`.`os_index` AS
`os_index`,`t_intf`.`os_name` AS `os_name`,`t_intf`.`bytes_in_s` AS
`bytes_in_s`,`t_intf`.`bytes_out_s` AS
`bytes_out_s`,`t_intf`.`bytes_in_udp` AS
```

```
`bytes_in_udp`,`t_intf`.`tcp_flow_count` AS `tcp_flow_count`,(select
`v_node`.`id` from `v_node` where ((`v_node`.`dpid` = `t_intf`.`dpid`) or
(`v_node`.`id` = `t_intf`.`id`))) AS `node_id` from `t_intf`) */;
/*View structure for view v_link */
/*!50001 DROP TABLE IF EXISTS `v_link` */;
/*!50001 CREATE ALGORITHM=UNDEFINED DEFINER=`tarom`@`%` SQL SECURITY
DEFINER VIEW `v_link` AS select `t_link`.`src_intf_id` AS
`src_intf_id`,`t_link`.`dst_intf_id` AS `dst_intf_id`,`t_link`.`bw` AS
`bw`,`v_intf`.`port_no` AS `src_intf_port_no`,`v_intf`.`name` AS
`src_intf_name`,`v_intf`.`os_index` AS
`src_intf_os_index`,`v_intf`.`os_name` AS
`src_intf_os_name`,`v_intf`.`bytes_in_s` AS
`src_bytes_in_s`,`v_intf`.`bytes_out_s` AS
`src_bytes_out_s`,`v_intf`.`bytes_in_udp` AS
`src_bytes_in_udp`,`v_intf`.`tcp_flow_count` AS
`src_tcp_flow_count`,`v_node`.`type` AS `src_node_type`,`v_node`.`dpid`
AS `src_node_dpid`,`v_intf`.`node_id` AS
`src_node_id`,`v_node`.`sflow_agent` AS
`src_node_sflow_agent`,`v_node`.`ipv4` AS
`src_node_ipv4`,`v_node`.`label` AS `src_node_label`,`v_intf_1`.`port_no`
AS `dst_intf_port_no`,`v_intf_1`.`name` AS
`dst_intf_name`,`v_intf_1`.`os_index` AS
`dst_intf_os_index`,`v_intf_1`.`os_name` AS
`dst_intf_os_name`,`v_intf_1`.`bytes_in_s` AS
`dst_bytes_in_s`,`v_intf_1`.`bytes_out_s` AS
`dst_bytes_out_s`,`v_intf_1`.`bytes_in_udp` AS
`dst_bytes_in_udp`,`v_intf_1`.`tcp_flow_count` AS
`dst_tcp_flow_count`,`v_node_1`.`type` AS
`dst_node_type`,`v_node_1`.`dpid` AS `dst_node_dpid`,`v_intf_1`.`node_id`
AS `dst_node_id`,`v_node_1`.`sflow_agent` AS
`dst_node_sflow_agent`,`v_node_1`.`ipv4` AS
`dst_node_ipv4`,`v_node_1`.`label` AS `dst_node_label` from ((((`t_link`
join `v_intf` on((`t_link`.`src_intf_id` = `v_intf`.`id`))) join `v_intf`
`v_intf_1` on((`t_link`.`dst_intf_id` = `v_intf_1`.`id`))) left join
`v_node` on((`v_intf`.`node_id` = `v_node`.`id`))) left join `v_node`
`v_node_1` on((`v_intf_1`.`node_id` = `v_node_1`.`id`))) */;
/*View structure for view v_node */
/*!50001 DROP TABLE IF EXISTS `v_node` */;
/*!50001 CREATE ALGORITHM=UNDEFINED DEFINER=`tarom`@`%` SQL SECURITY
DEFINER VIEW `v_node` AS (select `t_switch`.`id` AS `id`,'SWITCH' AS
`type`,NULL AS `hw_addr`,`t_switch`.`dpid` AS
`dpid`,`t_switch`.`sflow_agent` AS `sflow_agent`,`t_switch`.`sflow_agent`
AS `ipv4`,`t_switch`.`label` AS `label` from `t_switch`) union (select
`t_host`.`id` AS `id`,'HOST' AS `type`,`t_host`.`hw_addr` AS
`hw_addr`,NULL AS `dpid`,NULL AS `sflow_agent`,`t_host`.`ipv4` AS
`ipv4`,`t_host`.`label` AS `label` from `t_host`) */;
/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
```